Analyser et encadrer les erreurs dues à l'arithmétique flottante

Claude-Pierre Jeannerod, Nathalie Revol

Chapitre 5

Analyser et encadrer les erreurs dues à l'arithmétique flottante

Claude-Pierre Jeannerod
Nathalie Revol

L'arithmétique à virgule flottante est le moyen le plus couramment utilisé pour calculer avec des (approximations des) réels sur ordinateur. Si cette arithmétique est par nature inexacte, la norme IEEE 754 qui la régit définit un cadre strict dans lequel il est possible d'analyser et prédire de façon tout à fait rigoureuse le comportement de nombreux algorithmes numériques de base. Dans ce cours, nous illustrerons ce principe à l'aide de résultats obtenus très récemment, qui revisitent une partie de l'analyse numérique classique. Nous verrons aussi comment majorer de façon automatique les erreurs dues à l'arithmétique flottante et présenterons quelques résultats expérimentaux : l'approche présentée utilise l'arithmétique par intervalles. Comme l'arithmétique par intervalles est implantée en utilisant l'arithmétique flottante, nous détaillerons comment l'implantation tient compte des erreurs qui se produisent dans ses calculs internes.

5.1 Introduction

Précision finie et qualité numérique. L'arithmétique à virgule flottante est le moyen le plus couramment utilisé pour calculer avec des réels sur ordinateur, et ce notamment grâce à la spécification rigoureuse fournie par la norme IEEE 754 [18]. Du fait que cette arithmétique n'utilise qu'une précision finie (c'est-à-dire, un nombre de chiffres fixé à l'avance), chaque opération élémentaire comme $+$ ou $\times$ est susceptible de commettre une

petite erreur, appelée *erreur d'arrondi*. De plus, lors d'un calcul impliquant plusieurs opérations élémentaires, ces erreurs auront en général tendance à s'accumuler ou l'une d'entre elles pourra être amplifiée. Une question importante est donc de comprendre l'effet de ces erreurs sur la *qualité* du résultat renvoyé.

Dans ce cours, nous verrons deux façons d'aborder cette question, l'analyse dite *a priori* et une analyse automatique, *a posteriori*, et nous donnerons un aperçu des évolutions récentes de chacun de ces deux domaines.

Analyse a priori. Cette approche vise à analyser la façon dont un algorithme se comporte en arithmétique à virgule flottante, et ce quelles que soient les valeurs des données en entrée. Idéalement, on cherche à garantir a priori que la solution calculée $\widehat{s}$ a une certaine qualité numérique. Cela peut signifier que $\widehat{s}$ est « proche » d'une solution exacte ou, ce qui peut s'avérer plus facile à établir, que $\widehat{s}$ est la solution exacte pour des données « proches ». Dans les deux cas, il s'agit d'établir des majorations fines et rigoureuses de l'erreur commise. Dans ce cours, l'accent sera mis sur le rôle joué par l'arithmétique dans l'obtention de telles bornes. Nous verrons en particulier comment exploiter les (nombreuses) propriétés des nombres flottants pour établir des bornes fines, concises et faciles à interpréter.

Analyse automatique, a posteriori. Ici, le calcul de bornes d'erreur s'effectue en même temps que le calcul étudié, avec un jeu de valeurs pour les entrées. Ces entrées peuvent éventuellement être données avec une incertitude initiale, l'analyse incorporera alors cette incertitude et les erreurs d'arrondi qui se produisent lors des calculs et produira un encadrement de l'erreur sur le résultat final. Le pire cas étant considéré pour chacune de ces erreurs prise indépendamment, l'encadrement obtenu est fréquemment trop pessimiste : nous verrons quelques pistes pour obvier à ce problème.

Plan du cours. La section 5.2 donne une présentation détaillée de l'arithmétique à virgule flottante. Nous insisterons surtout sur ses deux principaux ingrédients, l'ensemble des nombres flottants et la notion d'arrondi correct, ainsi que sur quelques unes des nombreuses propriétés qui en découlent.

Nous verrons ensuite en section 5.3 comment exploiter ces propriétés pour analyser a priori le comportement de divers algorithmes classiques. Après un rappel de l'approche la plus courante (l'analyse inverse de Wilkinson), nous verrons quelques améliorations récentes obtenues pour des problèmes de base comme la somme de n nombres, le produit de matrices, ou encore l'évaluation très précise d'expressions de la forme $ab + cd$.

Nous donnerons une présentation générale de l'arithmétique par intervalles, qui est l'approche mise en œuvre dans ce cours pour analyser automatiquement les erreurs, en section 5.4, puis nous verrons les détails

de son implantation utilisant l'arithmétique flottante. Nous l'avons déjà signalé, l'encadrement des erreurs peut être pessimiste. Une solution qui demande peu d'expertise consiste, sans rien modifier d'autre, à augmenter la précision de calcul. Nous montrerons les bénéfices d'un changement de représentation des intervalles, par leur centre et leur rayon plutôt que par leurs extrémités. Enfin, nous évoquerons deux variantes qui réduisent les problèmes provoqués par la « perte » de certaines propriétés algébriques.

Nous conclurons en section 5.5 avec des suggestions de lectures pour approfondir les notions et approches présentées ou pour les prolonger.

5.2 Arithmétique à virgule flottante

5.2.1 Nombres à virgule flottante

Informellement, il s'agit de nombres rationnels de la forme $M\beta^E$ où β est un entier positif fixé, appelé *base*, et où M et E sont deux entiers relatifs de taille bornée. Plus précisément, $|M| < \beta^p$ où p est un entier positif appelé *précision*, et $E + p - 1 \in [e_{\min}, e_{\max}]$ où $e_{\min}$ et $e_{\max}$ sont deux entiers définissant la *plage d'exposants*.

L'ensemble $\mathbb{F}$ de tels nombres, appelé *système à virgule flottante*, est donc un sous-ensemble très particulier de $\mathbb{Q}$, entièrement caractérisé par les paramètres entiers β, p, $e_{\min}$ et $e_{\max}$. Par exemple, parler de « double précision IEEE » revient à fixer $\beta = 2$, $p = 53$, $e_{\min} = -1022$ et $e_{\max} = 1023$.

L'objectif de ce cours étant d'étudier l'effet d'une précision finie sur la qualité des calculs, nous idéaliserons un peu en fixant $e_{\min} = -\infty$ et $e_{\max} = +\infty$ de façon à ne jamais devoir nous préoccuper d'éventuels dépassements de capacité au niveau de l'exposant. En particulier, l'absence de borne inférieure sur la valeur des exposants permet de représenter tout élément non nul de $\mathbb{F}$ de façon unique sous forme *normalisée*, c'est-à-dire avec une mantisse M telle que $\beta^{p-1} \leqslant |M|$. L'ensemble $\mathbb{F}$ des nombres à virgule flottante en base β et précision p pourra donc être décrit comme suit :

$$\mathbb{F} = \{0\} \cup \left\{ M\beta^E \,:\, M, E \in \mathbb{Z},\ \ \beta^{p-1} \leqslant |M| < \beta^p \right\}. \tag{5.1}$$

Hypothèses. Dans la suite de ce cours, nous supposerons que β est pair et que $p \geqslant 2$. Ces hypothèses nous permettront de simplifier certains énoncés et certaines preuves, sans pour autant exclure de cas pratique significatif. En particulier, la norme IEEE 754 suppose que $\beta \in \{2, 10\}$ et les formats classiques qu'elle spécifie (et qui sont ceux dont nous disposons aujourd'hui sur nos machines) vérifient tous $p \geqslant 11$.

Représentations alternatives. Selon les cas, il peut être utile de décrire les éléments de $\mathbb{F}$ autrement qu'avec (5.1). En posant $m = |M|/\beta^{p-1}$ et $e = E + p - 1$, on voit tout d'abord que tout $f \in \mathbb{F}_{\neq 0}$ est aussi de la forme

$$\pm m\beta^e, \qquad m = \sum_{i=0}^{p-1} m_i \beta^{-i} =: (m_0 \bullet m_1 \dots m_{p-1})_\beta \tag{5.2}$$

avec $m_i \in \{0, 1, \dots, \beta - 1\}$ pour tout i, $m_0 \neq 0$ et $e \in \mathbb{Z}$. Cette seconde représentation, bien qu'étant moins concise que la première (qui est exclusivement entière), a l'avantage de faire apparaître explicitement la virgule ainsi que les chiffres de la mantisse en base β ; en particulier, m_0 est appelé *chiffre le plus significatif* et m_{p-1} est appelé *chiffre le moins significatif*.

Une troisième représentation est obtenue en récrivant la mantisse fractionnaire m apparaissant dans (5.2) à l'aide de l'*unité d'erreur d'arrondi*

$$u = \frac{1}{2}\beta^{1-p}.$$

En effet, m prenant les valeurs $1, 1 + \beta^{1-p}, 1 + 2\beta^{1-p}, \dots$, on vérifie facilement que tout élément de $\mathbb{F}$ est de la forme

$$\pm(1 + 2ku)\beta^e \tag{5.3}$$

avec $k \in \{0, 1, \dots, (\beta - 1)\beta^{p-1} - 1\}$ et $e \in \mathbb{Z}$. Comme nous le verrons plus loin, l'unité d'erreur d'arrondi u joue un rôle central dans toute analyse a priori, et il peut s'avérer particulièrement pratique que cette quantité apparaisse d'emblée dans la façon d'écrire les éléments de $\mathbb{F}$.

Propriétés. L'ensemble $\mathbb{F}$ défini en (5.1) peut être vu comme une « grille de nombres structurés » qui, à ce titre, possède de nombreuses propriétés. Notons tout d'abord que si $f \in \mathbb{F}$ alors $-f \in \mathbb{F}$ (symétrie) et $f\beta^e \in \mathbb{F}$ pour tout $e \in \mathbb{Z}$ (auto-similarité). Par conséquent, nous serons souvent amenés à nous concentrer sur le sous-ensemble $\mathbb{F} \cap [1, \beta)$; en rappelant (5.3), nous voyons que ce sous-ensemble est de la forme

$$\mathbb{F} \cap [1, \beta) = \{1,\ 1 + 2u,\ 1 + 4u,\ \dots,\ \beta - 2u\}.$$

Plus généralement, tout sous-ensemble de la forme $\mathbb{F} \cap [\beta^e, \beta^{e+1}]$ avec $e \in \mathbb{Z}$ possède la double propriété suivante :

— il contient $(\beta - 1)\beta^{p-1} + 1$ éléments régulièrement espacés ;
— la distance entre deux éléments consécutifs est égale à

$$2u\beta^e. \tag{5.4}$$

Ce second point implique qu'une puissance entière de la base est plus éloignée (d'un facteur multiplicatif égal à β) de son successeur que de son prédécesseur dans $\mathbb{F}$. Par exemple, le voisinage de 1 a la forme suivante :

$$\ldots, 1 - \tfrac{4u}{\beta},\ 1 - \tfrac{2u}{\beta},\ 1,\ 1 + 2u,\ 1 + 4u, \ldots$$

Du fait de cette « régularité par morceaux » de l'ensemble $\mathbb{F}$, les analyses d'erreur seront parfois plus délicates au voisinage des puissances entières de la base.

Enfin, certaines propriétés de $\mathbb{F}$ s'expriment facilement à l'aide des notions d'ufp (*unit in the first place*) et d'ulp (*unit in the last place*), définies de la façon suivante : pour $f \in \mathbb{F}_{\neq 0}$ représenté par $f = \pm m\beta^e$ comme en (5.2),

$$\mathrm{ufp}(f) = \beta^e \qquad \text{et} \qquad \mathrm{ulp}(f) = \beta^{e-p+1}.$$

En d'autres termes, $\mathrm{ufp}(f)$ donne l'ordre de grandeur de f, et $\mathrm{ulp}(f)$ le « poids » de son chiffre le moins significatif. Tout comme les fonctions usuelles 'signe' et 'valeur absolue', ces fonctions ufp et ulp permettront de manipuler les éléments de $\mathbb{F}$ indépendamment de leur représentation. [1] Parmi les propriétés utiles pour cela, mentionnons le fait que ces deux fonctions sont croissantes, la relation $\mathrm{ulp}(f) = \mathrm{ufp}(f)\beta^{1-p}$, et l'implication

$$f \neq 0 \qquad \Rightarrow \qquad \mathrm{ulp}(f) \leqslant |f|/\beta^{p-1} < \beta\mathrm{ulp}(f). \tag{5.5}$$

De plus, en rappelant que les exposants E et e dans (5.1) et (5.2) vérifient $e = E + p - 1$, nous pouvons écrire $\mathrm{ulp}(f) = \beta^E$ et conclure en particulier que

$$f \in \mathrm{ulp}(f)\mathbb{Z}. \tag{5.6}$$

Cette propriété, qui signifie que *tout nombre à virgule flottante est un multiple entier de son ulp*, sera très utile pour analyser des situations de calcul exact, comme le théorème de Sterbenz ou certaines *transformations exactes* ; voir en section 5.2.5.

5.2.2 Fonctions d'arrondi

L'approximation d'un réel par un nombre flottant se fait à l'aide d'une fonction $\circ : \mathbb{R} \to \mathbb{F}$ appelée *fonction d'arrondi* et telle que

$$t \in \mathbb{F} \quad \Rightarrow \quad \circ(t) = t; \qquad t \leqslant t' \quad \Rightarrow \quad \circ(t) \leqslant \circ(t'). \tag{5.7}$$

1. La fonction ufp a été introduite dans [56] et est en fait définie plus généralement sur les réels par $\mathrm{ufp}(0) = 0$ et $\mathrm{ufp}(t) = \beta^{\lfloor \log_\beta |t| \rfloor}$ pour tout $t \in \mathbb{R}_{\neq 0}$.

Cette définition générale (5.7) suffit déjà à garantir les trois propriétés importantes suivantes, dont la vérification est immédiate :

$$\begin{aligned} \circ(t) = 0 \qquad &\Leftrightarrow \qquad t = 0; \\ \beta^e \leqslant |t| < \beta^{e+1}, \quad e \in \mathbb{Z} \qquad &\Rightarrow \qquad \beta^e \leqslant |\circ(t)| \leqslant \beta^{e+1}; \\ \operatorname{sign}\big(\circ(t)\big) &= \operatorname{sign}(t). \end{aligned}$$

En d'autres termes, arrondir préserve le signe et, à un facteur β près, l'ordre de grandeur.

Nous allons maintenant spécialiser cette définition générale de façon à définir différents types d'arrondi (« au plus proche », « vers zéro », etc.).

Arrondi « au plus proche ». C'est le type d'arrondi le plus utilisé en pratique ; il sera noté RN pour « rounding to nearest » et vérifie, par définition,

$$|t - \mathrm{RN}(t)| = \min_{f \in \mathbb{F}} |t - f| \quad \text{pour tout } t \in \mathbb{R}. \tag{5.8}$$

Cette propriété, qui implique (5.7), doit être complétée par une règle (« tie-breaking rule ») définissant $\mathrm{RN}(t)$ de façon unique pour chaque réel t situé à mi-chemin entre deux éléments de $\mathbb{F}$ consécutifs.

La règle la plus commune, spécifiée dès la première version de la norme IEEE 754 en 1985, consiste à choisir le nombre flottant dont la mantisse entière M est paire. Par exemple, cette règle impose d'arrondir $t = 1 + u$ vers 1 puisque les deux nombres flottants qui encadrent t sont $f_1 = 1$ et $f_2 = 1 + 2u$ et que leurs mantisses entières sont $M_1 = \beta^{p-1}$ et $M_2 = \beta^{p-1} + 1$. La révision de 2008 de la norme IEEE 754 a introduit une seconde règle, selon laquelle le flottant retourné est celui de plus grande valeur absolue. Avec cette seconde règle (destinée surtout au calcul financier en base 10), l'arrondi au plus proche de $1 + u$ n'est plus 1, mais $1 + 2u$.

Dans la suite de ce cours, nous n'aurons pas à nous restreindre à ces deux règles, propres à la norme IEEE 754 ; nous pourrons plus généralement considérer n'importe quelle règle « raisonnable » qui, comme celles-ci, ne dépend ni du signe ni de l'ordre de grandeur du réel à arrondir. Cela reviendra à supposer que, pour tout $t \in \mathbb{R}$, la fonction RN vérifie

$$\mathrm{RN}(-t) = -\mathrm{RN}(t) \qquad \text{et} \qquad \mathrm{RN}(t\beta^k) = \mathrm{RN}(t)\beta^k, \quad \forall k \in \mathbb{Z}.$$

Ces hypothèses reflètent bien la symétrie et la structure de l'ensemble $\mathbb{F}$ et pourront, elles aussi, être exploitées pour simplifier certaines analyses.

Arrondis dirigés. La norme IEEE 754 spécifie aussi trois sortes d'arrondi dirigé, que nous noterons RD, RU et RZ pour « rounding down », « rounding

up » et « rounding to zero ». Ces fonctions sont définies à partir de (5.7) et de l'une des trois inégalités suivantes, valables pour tout $t \in \mathbb{R}$:

$$\mathrm{RD}(t) \leqslant t, \qquad t \leqslant \mathrm{RU}(t), \qquad |\mathrm{RZ}(t)| \leqslant |t|.$$

Cela signifie que $\mathrm{RD}(t)$ est le plus grand élément de $\mathbb{F}$ inférieur ou égal à t, que $\mathrm{RU}(t)$ est le plus petit élément de $\mathbb{F}$ supérieur ou égal à t, et que RZ tronque t pour n'en garder que les p chiffres (en base β) les plus significatifs.

Comme pour RN, nous avons $\circ(t\beta^k) = \circ(t)\beta^k$ pour tout $k \in \mathbb{Z}$ et $\circ \in \{\mathrm{RD}, \mathrm{RU}, \mathrm{RZ}\}$, ainsi que $\mathrm{RZ}(-t) = -\mathrm{RZ}(t)$. Cette dernière propriété n'est en revanche pas satisfaite par RU et RD, pour lesquels nous avons la relation $\mathrm{RD}(-t) = -\mathrm{RU}(t)$.

Notons enfin que si $t \notin \mathbb{F}$ alors $\mathrm{RD}(t)$ et $\mathrm{RU}(t)$ définissent l'unique paire d'éléments consécutifs de $\mathbb{F}$ telle que $t \in [\mathrm{RD}(t), \mathrm{RU}(t)]$.

Bornes d'erreur. Lorsque t n'est pas un flottant, lui appliquer une fonction d'arrondi $\circ$ entraîne systématiquement une erreur, appelée *erreur d'arrondi* et que l'on peut définir de façon absolue comme $|t - \circ(t)|$ ou de façon relative à l'aide des deux fonctions de $\mathbb{R}_{\neq 0}$ dans $\mathbb{R}$ suivantes :

$$E_1(t) = \frac{|t - \circ(t)|}{|t|} \qquad \text{et} \qquad E_2(t) = \frac{|t - \circ(t)|}{|\circ(t)|}.$$

Voyons maintenant comment majorer ces trois erreurs lorsque $\circ = \mathrm{RN}$. En notant e l'entier tel que $\beta^e \leqslant |t| < \beta^{e+1}$ et en rappelant que la distance entre deux éléments consécutifs de $\mathbb{F} \cap [\beta^e, \beta^{e+1}]$ vaut $2u\beta^e$, nous voyons que l'erreur absolue commise en arrondissant « au plus proche » vérifie

$$|t - \mathrm{RN}(t)| \leqslant \frac{1}{2} \times 2u\beta^e = u\beta^e. \tag{5.9}$$

Cette borne dépend de l'unité d'erreur d'arrondi u mais aussi de l'ordre de grandeur du réel t. Puisque $\beta^e \leqslant |t|$ implique $\beta^e \leqslant |\mathrm{RN}(t)|$, nous déduisons ensuite que $E_2(t) \leqslant u$. Cela vaut également pour $E_1(t)$, mais dans ce cas il est possible de faire légèrement mieux : en effet,

— si $|t| \geqslant (1+u)\beta^e$ alors $E_1(t) \leqslant u\beta^e/|t| \leqslant u/(1+u)$;
— si $|t| < (1+u)\beta^e$ alors $|\mathrm{RN}(t)| = \beta^e$ et, puisque $\mathrm{RN}(t)$ et t ont le même signe, $E_1(t) = 1 - \beta^e/|t| < 1 - 1/(1+u) = u/(1+u)$.

Nous venons donc de démontrer le résultat suivant :

Théorème 5.2.1. *Pour tout réel t non nul, les erreurs relatives commises en l'arrondissant « au plus proche » vérifient*

$$E_1(t) \leqslant \frac{u}{1+u} \qquad \textit{et} \qquad E_2(t) \leqslant u.$$

L'inégalité $E_1(t) \leqslant u/(1+u)$ peut être attribuée à Dekker [12] et Knuth [27, p. 232], et l'inégalité $E_2(t) \leqslant u$ apparaît dans [16, p. 39]. Ces deux bornes sont atteintes par exemple pour $t = 1 + u$ et avec la règle d'arrondi « au plus proche vers le flottant de mantisse paire ». Leur principal intérêt est de majorer l'erreur d'arrondi indépendamment de la valeur du réel à arrondir, par une quantité définie exclusivement par la base et la précision. Par exemple, en base 2 nous avons $u = 2^{-p}$ et donc

$$\frac{u}{1+u} \approx u \approx \begin{cases} 5.9 \times 10^{-8} & \text{si } p = 24 \quad \text{(simple précision)}, \\ 1.1 \times 10^{-16} & \text{si } p = 53 \quad \text{(double précision)}. \end{cases}$$

Pour les arrondis dirigés, il suffit essentiellement de remplacer u par $2u$ dans les trois bornes précédentes. Plus précisément, l'analogue de (5.9) pour $\circ \in \{\mathrm{RD}, \mathrm{RU}, \mathrm{RZ}\}$ est l'inégalité stricte $|t - \circ(t)| < 2u\beta^e$ et, concernant les erreurs relatives, on vérifie facilement que $E_1(t) < 2u/(1+2u)$ et $E_2(t) < 2u$ lorsque $\circ = \mathrm{RZ}$, et que $E_1(t), E_2(t) < 2u$ lorsque $\circ \in \{\mathrm{RD}, \mathrm{RU}\}$.

5.2.3 Arrondi correct des opérations élémentaires

Le résultat exact d'une opération élémentaire $\mathrm{op} \in \{+, -, \times, /\}$ sur des éléments de $\mathbb{F}$ n'est en général pas dans $\mathbb{F}$. Pour une fonction d'arrondi $\circ$ donnée, l'arithmétique à virgule flottante va alors consister à approcher le résultat exact $r = x \,\mathrm{op}\, y$ par la valeur $\widehat{r} \in \mathbb{F}$ de son arrondi :

$$\widehat{r} := \circ(x \,\mathrm{op}\, y). \tag{5.10}$$

Par exemple, si $x = y = \beta^p - 1$, le produit $xy = \beta^{2p} - 2\beta^p + 1$ est un entier ayant $2p$ chiffres en base β (soit le double de la précision p offerte par $\mathbb{F}$) :

$$xy = (\underbrace{* * * \ldots *}_{p-1} \bar{*} \underbrace{000 \ldots 0}_{p-1} 1)_\beta, \qquad * = \beta - 1 \neq 0, \qquad \bar{*} = \beta - 2.$$

Ainsi, $xy \notin \mathbb{F}$ et, en arithmétique à virgule flottante avec arrondi au plus proche, le résultat de cette multiplication est $\widehat{r} = \mathrm{RN}(xy) = \beta^{2p} - 2\beta^p = (\beta^p - 2)\beta^p \in \mathbb{F}$.

La méthode de calcul définie par (5.10) est appelée *arrondi correct* et peut être vue comme un algorithme, un mécanisme de composition de fonctions permettant de contrôler à chaque opération l'augmentation éventuelle de la taille du résultat exact [62]. La norme IEEE 754 exige l'arrondi correct pour les quatre opérations de base, la racine carrée et l'opération $(x, y, z) \mapsto xy + z$ (appelée FMA pour « fused multiply-add »). Dans tous les cas, $\circ \in \{\mathrm{RN}, \mathrm{RD}, \mathrm{RU}, \mathrm{RZ}\}$ avec, pour RN, les deux règles évoquées en section 5.2.2 pour départager les cas d'égalité.

Grâce à (5.10), le résultat d'une opération en virgule flottante est donc défini sans aucune ambiguïté. En pratique, la norme IEEE 754 spécifie aussi le résultat à retourner en cas de dépassement de capacité (la plage d'exposants étant bornée) ou en cas d'opération illicite (comme $0/0$ ou $\sqrt{-1}$), et ce grâce aux notions d'infinis et de NaN (Not-a-Number). Dans ce cours, nous nous contenterons d'exploiter (5.10), mais il faut garder à l'esprit que l'implantation efficace de cette règle ainsi que de tous les autres aspects de la norme IEEE 754 est un domaine de recherche en soi (représenté notamment par la série de conférences ARITH [17]).

Disposer d'une arithmétique avec arrondis dirigés, et en particulier RD et RU, permet de calculer les deux flottants consécutifs qui encadrent le résultat exact (si celui-ci n'est pas lui-même un flottant) :

$$x \operatorname{op} y \in [\mathrm{RD}(x \operatorname{op} y), \mathrm{RU}(x \operatorname{op} y)].$$

Comme nous le verrons en section 5.4, cela sera très pratique pour implanter les arithmétiques par intervalles.

Modélisation standard. En appliquant à $t = x \operatorname{op} y$ les bornes d'erreur du théorème 5.2.1, nous voyons que pour l'arrondi au plus proche la valeur calculée $\widehat{r} = \mathrm{RN}(x \operatorname{op} y)$ est de la forme

$$\widehat{r} = (x \operatorname{op} y)(1+\delta_1), \qquad |\delta_1| \leqslant \frac{u}{1+u}; \tag{5.11}$$

$$= \frac{x \operatorname{op} y}{1+\delta_2}, \qquad |\delta_2| \leqslant u. \tag{5.12}$$

Pour les arrondis dirigés, la discussion en fin de section 5.2.2 montre qu'il suffit de remplacer les deux inégalités ci-dessus par $|\delta_1|, |\delta_2| < 2u$. (Dans tous les cas, δ_1 et δ_2 désignent des réels dont la valeur est inconnue a priori.)

Les relations (5.11) et (5.12) sont appelées *modèles standard de l'arithmétique à virgule flottante* et expriment la principale conséquence de l'arrondi correct : les erreurs relatives commises par chaque opération élémentaire sont majorées par une « petite constante » exprimable exclusivement à l'aide de l'unité d'erreur d'arrondi u. En pratique, ces modèles sont valables pour l'arithmétique IEEE 754 dès lors que l'opération en question n'entraîne aucun dépassement de capacité.

La plupart des analyses d'erreur utilisent en fait une version simplifiée du premier modèle standard, selon laquelle $|\delta_1| < u$. Cependant, la borne $|\delta_1| \leqslant u/(1+u)$, bien que n'étant que très légèrement plus fine, permet de simplifier considérablement certaines démonstrations [23]. En base 2, le second modèle standard fournit une borne sur l'erreur absolue qui est à la fois atteignable et représentable : $|\widehat{r} - x \operatorname{op} y| \leqslant b$ avec $b = \widehat{r}\, 2^{-p} \in \mathbb{F}$.

Plus généralement, ce second modèle permet de démontrer la validité de certaines analyses a posteriori (« running error analysis » [16, §3.3]) et, en combinaison avec le premier modèle, d'obtenir des bornes d'erreur très lisibles pour des calculs spécifiques mais importants comme la résolution d'un système linéaire triangulaire [16, §8.1].

Les modèles (5.11) et (5.12) ne reflètent bien évidemment pas toutes les propriétés contenues dans la règle d'arrondi correct (5.10) : ils ignorent notamment la structure des éléments de $\mathbb{F}$ décrite en (5.3) ainsi que des propriétés importantes des fonctions d'arrondi comme celles données en (5.7). Lors d'une analyse d'erreur a priori, l'idéal sera donc souvent d'utiliser ces propriétés « de bas niveau » en complément des modèles standard.

5.2.4 Quelques surprises typiques

Influence de la base. Notons tout d'abord que puisque les flottants sont des rationnels dont le dénominateur est une puissance entière de la base, une constante aussi simple que $1/10$ n'est pas représentable exactement en base 2 (et ce quelle que soit la précision p).

Un autre exemple typique est celui du calcul de la moyenne de deux nombres : si pour $x, y \in \mathbb{F}$ on évalue $(x+y)/2$ de façon usuelle, le résultat obtenu est $\widehat{r} = \circ(\circ(x+y)/2)$. En base 2, cette valeur approchée $\widehat{r}$ vérifie ce qu'on attend d'une moyenne :

$$\min\{x, y\} \leqslant \widehat{r} \leqslant \max\{x, y\}.$$

Cette double inégalité est une conséquence de (5.7) et du fait que $2x, 2y \in \mathbb{F}$ si $\beta = 2$. En revanche, elle n'est plus vraie en base 10 ; voir [3, 58] pour des contre-exemples et des façons de pallier cette difficulté.

Perte de certaines propriétés algébriques. La commutativité de $+$, $-$, $\times$ est préservée, comme conséquence directe de (5.10). En revanche, associativité et distributivité (de $\times$ par rapport à $\pm$) ne sont plus garanties. Par exemple, si $x, y, z \in \mathbb{F}$ alors, en général, $\circ(\circ(x+y)+z) \neq \circ(x+\circ(y+z))$. Nous serons donc souvent amenés à considérer plusieurs schémas différents pour l'évaluation d'une même fonction. Même si, en principe, chaque schéma requiert une analyse d'erreur spécifique, nous verrons en section 5.3 que dans certains cas il est possible d'établir des bornes d'erreur valables indépendamment de l'ordre des opérations.

Élimination catastrophique. Même si chaque opération élémentaire bénéficie de l'arrondi correct et donc d'une erreur relative au plus u, en enchaîner deux peut suffire à produire un résultat dont l'erreur relative vaut 1—ce

qui signifie qu'aucun de ses chiffres ne peut être considéré comme correct. Ce phénomène, appelé *élimination catastrophique*, se produit par exemple pour $\widehat{r} = \mathrm{RN}(\mathrm{RN}(x+y)+z)$ avec $(x,y,z) = (1, u/\beta, -1) \in \mathbb{F}^3$. En effet, $\widehat{r} = 0$ et, la somme exacte r étant non nulle, l'erreur relative est alors $|\widehat{r}-r|/|r| = 1$. Cette perte quasi instantanée de toute qualité numérique est liée à la notion plus générale de *conditionnement* ; nous renvoyons à [7, 28] pour une présentation détaillée de ce concept.

Éviter ou retarder l'apparition de ce genre de phénomène peut requérir diverses modifications, de l'arithmétique ou du calcul lui-même : augmenter la précision p, récrire l'algorithme (comme remplacer $a^2 - b^2$ par $(a+b)(a-b)$, par exemple) ou encore introduire des techniques de *compensation* consistant à calculer certaines erreurs d'arrondi et à s'en servir pour améliorer la qualité du résultat initial [31]. Ce dernier aspect sera illustré en section 5.3.4 pour le calcul précis d'expressions de la forme $ab + cd$.

5.2.5 Opérations exactes et erreurs représentables

Nous terminons cette partie en présentant quelques situations où certaines quantités sont calculées de façon *exacte* en arithmétique à virgule flottante. Les propriétés ci-dessous peuvent s'avérer cruciales pour concevoir des algorithmes très précis et démontrer qu'ils le sont vraiment.

Opérations exactes. Si x et y sont dans $\mathbb{F}$, leur produit sera calculé exactement par (5.10) dès que le produit $M_x M_y$ de leurs mantisses entières est lui-même dans $\mathbb{F}$ (des cas particuliers étant le produit d'un flottant par une puissance entière de la base, et le produit d'entiers x et y tels que $|xy| \leqslant \beta^p$). L'analogue pour la soustraction est le théorème de Sterbenz [58, §4.3] :

Théorème 5.2.2. *Si $x, y \in \mathbb{F}$ sont tels que $y/2 \leqslant x \leqslant 2y$ alors $x - y \in \mathbb{F}$.*

Ce résultat, valable quelle que ce soit la base, nous dit que la différence de deux flottants suffisamment proches l'un de l'autre est elle-même un flottant et sera donc calculée exactement. On peut le démontrer très simplement grâce aux propriétés de $\mathbb{F}$ vues en section 5.2.1 : quitte à échanger x et y, nous pouvons supposer que $0 \leqslant y \leqslant x \leqslant 2y$. Si $y = 0$ alors $x - y = x \in \mathbb{F}$. Sinon, $0 < \mathrm{ulp}(y) \leqslant \mathrm{ulp}(x)$ et donc, d'après (5.6), $x - y \in \mathrm{ulp}(y)\mathbb{Z}$. Ainsi, $x - y = M\beta^E$ avec M et E deux entiers tels que $\beta^E = \mathrm{ulp}(y)$ et, d'après (5.5), $|M| = (x-y)/\mathrm{ulp}(y) \leqslant y/\mathrm{ulp}(y) < \beta^p$.

Erreurs représentables et transformations exactes associées. Une autre conséquence remarquable de la règle d'arrondi correct (5.10) est l'implication suivante, selon laquelle l'erreur absolue commise en additionnant ou

multipliant deux flottants est elle-même un flottant :

$$x, y \in \mathbb{F}, \quad \mathrm{op} \in \{+, \times\} \qquad \Rightarrow \qquad x \,\mathrm{op}\, y - \mathrm{RN}(x \,\mathrm{op}\, y) \in \mathbb{F}.$$

Ce résultat se démontre de la même manière que le théorème 5.2.2 et, dans le cas de la multiplication, reste valable pour les arrondis dirigés.

Étant donnés x, y et $\widehat{r} = \mathrm{RN}(x \,\mathrm{op}\, y)$, on sait de plus calculer l'erreur correspondante $e = x \,\mathrm{op}\, y - \widehat{r}$ à l'aide de seulement quelques opérations élémentaires en précision p (et donc sans recourir à une précision étendue). Pour la multiplication, il suffit d'utiliser l'opération FMA de la façon suivante :

$$\widehat{r} := \mathrm{RN}(xy); \qquad e := \mathrm{RN}(xy - \widehat{r}). \tag{5.13}$$

Pour l'addition, Kahan [26] et Dekker [12] ont remarqué que deux opérations supplémentaires suffisent dès que $\beta \in \{2, 3\}$ et $|x| \geqslant |y|$: [2]

$$\widehat{r} := \mathrm{RN}(x + y); \qquad \widehat{y} := \mathrm{RN}(\widehat{r} - x); \qquad e := \mathrm{RN}(y - \widehat{y}).$$

(Très schématiquement, l'idée est que si $\widehat{r}$ contient $x +$ (la partie haute de y), alors $\widehat{y}$ contient la partie haute de y et e contient la partie basse de y.) Chacun de ces algorithmes définit une *transformation exacte* ou *EFT* (pour « error-free transform » [42]), qui à $(x, y) \in \mathbb{F}^2$ associe $(\widehat{r}, e) \in \mathbb{F}^2$ tel que $x \,\mathrm{op}\, y = \widehat{r} + e$.

5.3 Analyse d'erreur a priori

5.3.1 Approche classique : l'analyse inverse de Wilkinson

La façon la plus courante d'analyser a priori l'effet des erreurs d'arrondi sur une solution approchée $\widehat{s}$ est l'*analyse inverse*, popularisée par Wilkinson dès les années soixante [65] et fondée sur la conséquence suivante, très simple, du modèle standard (5.11) : pour $x, y \in \mathbb{F}$, l'approximation $\widehat{r} := \mathrm{RN}(x \,\mathrm{op}\, y)$ produite à chaque opération élémentaire est le *résultat exact de cette opération sur des données « proches »* $\widetilde{x}, \widetilde{y} \in \mathbb{R}$, c'est-à-dire

$$\widehat{r} = \widetilde{x} \,\mathrm{op}\, \widetilde{y}, \quad \widetilde{x} := x(1 + \delta_x), \quad \widetilde{y} := y(1 + \delta_y), \quad |\delta_x|, |\delta_y| \leqslant u. \tag{5.14}$$

Par exemple, on déduit de (5.11) que $\mathrm{RN}(x + y)$ est la somme exacte de $x(1 + \delta_1)$ et $y(1 + \delta_1)$. Dans le cas de la multiplication, on peut voir $\mathrm{RN}(xy)$ comme le produit exact de x et $y(1 + \delta_1)$, ou de $x(1 + \delta_1)$ et y, ou encore, si cela s'avère plus pratique, de $x(1 + \delta_1)^{1/2}$ et $y(1 + \delta_1)^{1/2}$.

2. Knuth [27] et Møller [33] ont montré que cinq additions supplémentaires (au lieu de deux) suffisent à s'affranchir de cette double condition.

Principe. Une analyse inverse consiste à appliquer (5.14) à chacune des opérations élémentaires utilisées pour produire $\widehat{s}$ et, plutôt que chercher à montrer que l'écart entre cette solution approchée $\widehat{s}$ et la solution exacte du problème est « petit » (ce qui peut s'avérer difficile, voire impossible), elle cherche à

— exprimer $\widehat{s}$ comme une solution exacte du même problème pour des données perturbées, et à
— majorer la taille, appelée *erreur inverse*, des plus petites perturbations possibles (pour une norme donnée).

Schématiquement, cette approche permet de considérer que commettre des erreurs d'arrondi équivaut à perturber les données du problème et, ainsi, de simplifier l'analyse numérique d'un algorithme en la ramenant à une analyse de la sensibilité du problème auquel l'algorithme s'applique. De plus, dans le cas de données inexactes (ce qui arrive souvent en pratique), si l'erreur inverse a le même ordre de grandeur que cette incertitude sur les données, on peut alors considérer que la solution calculée $\widehat{s}$ est tout à fait satisfaisante : elle pourrait être la solution exacte pour les données exactes.

Exemples. Étant donnés $x_1, \dots, x_n \in \mathbb{F}$, considérons tout d'abord l'évaluation du produit $s = \prod_{i=1}^{n} x_i$ à l'aide de $n-1$ multiplications. On montre facilement par récurrence sur n que, pour tout ordre d'évaluation, il existe $n-1$ réels δ_i tels que la valeur calculée vérifie $\widehat{s} = s(1+\theta_{n-1})$ avec

$$1+\theta_{n-1} := \prod_{i=1}^{n-1}(1+\delta_i), \qquad |\delta_i| \leqslant u. \tag{5.15}$$

Dans ce cas, la situation est idéale au sens où à la fois l'*erreur directe* (c'est-à-dire l'écart relatif entre la solution approchée et la solution exacte) et l'erreur inverse sont, pour n fixé et $u \to 0$, toujours en $O(u)$. En effet, d'une part nous déduisons de (5.15) que quelles que soient les valeurs des x_i,

$$\begin{aligned} |\widehat{s}-s| \leqslant \alpha|s|, \qquad \alpha &:= (1+u)^{n-1} - 1 \\ &= (n-1)u + O(u^2); \end{aligned} \tag{5.16}$$

d'autre part, $\widehat{s}$ se récrit aussi comme le produit exact de n réels $\widetilde{x}_i$, définis par exemple par $\widetilde{x}_1 = x_1$ et $\widetilde{x}_i = x_i(1+\delta_{i-1})$ pour $i \geqslant 2$, et donc obtenus par des perturbations relatives des x_i d'au plus u.

Considérons maintenant le cas, plus délicat, de l'évaluation de la somme $s = \sum_{i=1}^{n} x_i$. L'exemple d'élimination catastrophique donné en section 5.2.4 indique qu'on ne peut plus espérer majorer l'erreur directe comme en (5.16) pour tous les x_i. Cependant, nous allons voir que pour n fixé l'erreur inverse est encore en $O(u)$. Par exemple, si l'évaluation se fait

« de gauche à droite » selon le schéma $(\dots((x_1+x_2)+x_3)+\cdots)+x_n$, on montre facilement en appliquant $n-1$ fois (5.14) que la valeur obtenue s'écrit

$$\widehat{s} = (x_1+x_2)(1+\theta_{n-1}) + \sum_{i=3}^{n} x_i(1+\theta_{n-i+1}),$$

chaque θ_i étant une expression de la forme (5.15). Par conséquent, $\widehat{s}$ est la somme exacte des n réels $\widetilde{x}_i$ tels que $\widetilde{x}_1 = x_1(1+\theta_{n-1})$ et $\widetilde{x}_i = x_i(1+\theta_{n-i+1})$ pour $i \geqslant 2$, et on déduit que l'erreur inverse est majorée par $\max_i |\theta_i| \leqslant \alpha$ avec α défini en (5.16). En procédant par récurrence sur n, on obtient cette conclusion indépendamment de l'ordre d'évaluation de la somme.

L'analyse inverse que nous venons de faire pour la somme de n nombres flottants permet, par simple application de l'inégalité triangulaire, de majorer l'erreur directe de la façon suivante : si $s \neq 0$ alors

$$\frac{|\widehat{s}-s|}{|s|} \leqslant \alpha\, C, \qquad C := \frac{\sum_{i=1}^{n}|x_i|}{|\sum_{i=1}^{n} x_i|}. \tag{5.17}$$

Le nombre C, appelé *conditionnement*, dépend du problème (sommer n nombres) et de ses données (les x_i), mais ne dépend pas de l'algorithme utilisé (la façon de « parenthéser » la somme des x_i). Si $C = O(1)$ alors nous avons la garantie a priori que l'erreur directe sera petite (de l'ordre de u pour n fixé) et qu'ainsi $\widehat{s}$ sera une excellente approximation de la somme exacte. En revanche, si $C \gtrsim 1/u$ alors $\alpha\, C \gtrsim 1$, indiquant le risque que la valeur calculée $\widehat{s}$ soit totalement fausse ; c'est précisément ce qui se produit lors de l'élimination catastrophique en section 5.2.4, pour laquelle $\alpha \approx 2u$ et $C \approx 2\beta/u$.

Avantages et inconvénients. L'analyse inverse a été développée bien au-delà des deux exemples très simples que nous venons de voir et, comme en témoignent les 600 et quelques pages du livre de Higham [16], cette approche s'est révélée d'une extrême utilité pour prédire et expliquer le comportement de nombreux algorithmes, notamment en algèbre linéaire.

Sur le plan de l'arithmétique flottante, l'analyse inverse a l'avantage d'une certaine simplicité en ne faisant appel, la plupart du temps, qu'aux modèles standard (la complexité des analyses venant plutôt des techniques d'analyse matricielle, parfois sophistiquées, mises en œuvre pour réussir à rendre les bornes d'erreur suffisamment fines et lisibles).

Bien sûr, se limiter aux modèles (5.11) et (5.12) signifie aussi que l'on se prive des nombreuses propriétés « de bas niveau » qu'implique la règle d'arrondi correct (5.10). Dans les sous-sections qui suivent, nous allons voir que la prise en compte de (certaines de) ces propriétés permet au moins deux choses : d'une part, obtenir des bornes plus fines et plus concises que,

par exemple, celle en (5.16) ; d'autre part, prouver que certains algorithmes sont toujours très précis.

5.3.2 Somme de n nombres flottants

Reprenons ici l'exemple de l'évaluation de $s = \sum_{i=1}^{n} x_i$ pour $x_i \in \mathbb{F}$. La borne d'erreur (5.17) est facile à obtenir, facile à interpréter et valable quel que soit l'ordre d'évaluation ; de plus, il existe des x_i pour lesquels le ratio erreur/(borne d'erreur) tend vers 1 lorsque $u \to 0$. (Prendre par exemple $x_1 = 1$ et $x_i = u$ pour $i \geqslant 2$.) Le seul défaut de cette borne concerne le terme α, dont la forme initiale $(1+u)^{n-1} - 1$ ne sera pas facile à manipuler lors d'analyses de plus haut niveau et dont la réécriture $(n-1)u + O(u^2)$ ne permet pas une majoration rigoureuse. (Notons aussi que remplacer u par $u/(1+u)$ ne suffit pas à supprimer les termes en $O(u^2)$ lorsque $n \geqslant 5$.)

Notation de Higham. Un compromis désormais classique consiste à utiliser la notation suivante, utilisée systématiquement par Higham dans [16] :

$$\gamma_k := \frac{ku}{1-ku} \qquad \text{si } ku < 1.$$

Ainsi, on pourra remplacer α par la borne γ_{n-1}, à la fois concise, rigoureuse et facile à propager grâce à des inégalités comme $\gamma_k + \gamma_\ell + \gamma_{k\ell} \leqslant \gamma_{k+\ell}$. Notons cependant que γ_{n-1} est encore de la forme $(n-1)u + O(u^2)$ et que la dimension n doit cette fois vérifier $(n-1)u < 1$.

Borne de Rump. Plus récemment, il a été observé par Rump [53] que pour la sommation « de gauche à droite », l'expression $\alpha = (1+u)^{n-1} - 1$ peut en fait toujours être remplacée par

$$\alpha = (n-1)u.$$

Cela conduit à une borne d'erreur plus fine et plus simple, sans terme quadratique en u et sans condition sur la valeur de n.

Pour établir ce résultat, Rump utilise non seulement le modèle standard (5.11), mais aussi la propriété suivante, qui découle directement de la définition d'arrondi « au plus proche » (5.8) appliquée à la somme de deux nombres flottants :

$$x, y \in \mathbb{F} \quad \Rightarrow \quad |\mathrm{RN}(x+y) - (x+y)| \leqslant \min\{|x|, |y|\}.$$

Plus précisément, cette propriété est exploitée dans une récurrence sur n visant à majorer l'erreur directe $|\widehat{s} - \sum_{i=1}^{n} x_i|$ par $(n-1)u \sum_{i=1}^{n} |x_i|$: on l'utilise si $|x_n| \leqslant u \sum_{i<n} |x_i|$, l'autre cas étant traité par le modèle standard.

Une généralisation de cette approche à tout ordre d'évaluation a été proposée dans [22, §3], permettant d'aboutir au résultat général suivant.

Théorème 5.3.1. *Soient $x_1, \ldots, x_n \in \mathbb{F}$. Si $\widehat{s} \in \mathbb{F}$ est le résultat de l'évaluation de $\sum_{i=1}^n x_i$ en arrondi « au plus proche » et pour un ordre quelconque, alors*

$$\left|\widehat{s} - \sum_{i=1}^n x_i\right| \leqslant (n-1)u \sum_{i=1}^n |x_i|.$$

5.3.3 Briques de base pour l'algèbre linéaire et les polynômes

Des améliorations similaires à celle présentée dans le théorème 5.3.1 ont été obtenues pour plusieurs autres problèmes de base, listés dans la table 5.1. Les algorithmes auxquels ces bornes d'erreur s'appliquent sont les schémas classiques (décrits notamment dans [16]), et la signification de α dépend du problème : par exemple, pour le produit de matrices, α est tel que $|\widehat{C} - AB| \leqslant \alpha |A||B|$ avec $A \in \mathbb{F}^{*\times n}$ et $B \in \mathbb{F}^{n\times *}$, $\widehat{C}$ désignant la matrice effectivement calculée, et l'inégalité et les valeurs absolues s'appliquant à chaque coefficient des matrices concernées ; pour la résolution de systèmes triangulaires et les factorisations LU et de Cholesky, nous considérons les majorations d'erreur inverse usuelles $|\Delta T| \leqslant \alpha |T|$ pour $(T + \Delta T)\widehat{x} = b$, $|\Delta A| \leqslant \alpha |\widehat{L}||\widehat{U}|$ pour $\widehat{L}\widehat{U} = A + \Delta A$, et $|\Delta A| \leqslant \alpha |\widehat{R}^T||\widehat{R}|$ pour $\widehat{R}^T\widehat{R} = A + \Delta A$. (Les matrices T, $\widehat{U}$, $\widehat{R}$ sont dans $\mathbb{F}^{n\times n}$ et $\widehat{L} \in \mathbb{F}^{m\times n}$ avec $m \geqslant n$.)

TABLE 5.1 – *Améliorations récentes de quelques bornes classiques. Sauf indication contraire, l'ordre des opérations n'est pas spécifié et ($\star$) signifie « si $n \lesssim u^{-1/2}$ ».*

Problème	Valeur de α		Référence(s)
prod. scalaire, prod. de matrices	nu		[22]
norme euclidienne $(\sum_{i=1}^n x_i^2)^{1/2}$	$(\frac{n}{2}+1)u$		[23]
$Tx = b, \quad A = LU$	nu		[55]
$A = R^T R$	$(n+1)u$		[55]
produit $\prod_{i=1}^n x_i$	$(n-1)u$	($\star$)	[54]
schéma de Horner pour $\sum_{i=0}^n a_i x^i$	$2nu$	($\star$)	[54]

Notons que les expressions de α données dans la table 5.1 n'ont aucun terme en $O(u^2)$, contrairement à celles obtenues avec l'approche classique présentée en section 5.3.1. Dans les deux cas ($\star$), le prix à payer pour ces bornes plus simples et plus fines est une condition (nécessaire) sur n détaillée dans [54].

5.3.4 Algorithmes précis pour évaluer $ab + cd$

Considérons maintenant l'évaluation de $ab + cd$ pour $a, b, c, d \in \mathbb{F}$. Cette opération, qui a de nombreuses applications (arithmétique complexe, discriminants, prédicats d'orientation robustes en géométrie algorithmique, ...), ne fait pas partie des fonctions pour lesquelles la norme IEEE 754 requiert ou recommande l'arrondi correct. Elle est donc susceptible d'être implantée surtout en logiciel, à l'aide d'opérations élémentaires comme $+$, $\times$ ou FMA. Pour ce faire, il faut cependant être un peu soigneux : en effet, un schéma classique comme $\mathrm{RN}(\mathrm{RN}(ab) + \mathrm{RN}(cd))$ ou, si un FMA est disponible, $\mathrm{RN}(ab + \mathrm{RN}(cd))$ peut conduire à une « élimination catastrophique » et ainsi donner un résultat totalement faux.

Algorithme de Kahan. Pour éviter ce genre de désagrément, Kahan a proposé l'algorithme suivant (décrit dans [16, p. 60]) :

$$\widehat{w} := \mathrm{RN}(cd); \quad \widehat{f} := \mathrm{RN}(ab + \widehat{w}); \quad e := \mathrm{RN}(cd - \widehat{w}); \quad \widehat{r} := \mathrm{RN}(\widehat{f} + e).$$

Le FMA est utilisé une première fois pour approcher $ab + cd$ à l'aide de $\widehat{f}$. Sur les $2p$ chiffres du produit exact cd, les p chiffres les moins significatifs sont « perdus » par arrondi, de sorte que $\widehat{f}$ pourra fournir une très mauvaise approximation de $ab + cd$ lorsque $ab \approx -cd$. Le FMA est alors utilisé une seconde fois, pour implanter une EFT du produit cd comme en (5.13) et obtenir ainsi l'erreur $e = cd - \widehat{w}$ exactement ; cette erreur est enfin ajoutée à l'approximation initiale $\widehat{f}$ afin d'en améliorer la qualité.

L'algorithme de Kahan est toujours très précis au sens où l'erreur relative sur sa sortie est en $O(u)$, et nous allons voir que ce résultat s'obtient en combinant le modèle standard (5.11) et des propriétés « de bas niveau ».

Tout d'abord, on déduit facilement de $e = cd - \widehat{w}$ et de (5.11) que

$$|\widehat{r} - r| \leqslant 2u|r| + u|e|, \qquad r = ab + cd.$$

Le modèle standard implique de plus que $|e| \leqslant u|cd|$. Cependant, utiliser cette majoration pour $|e|$ conduit à la borne d'erreur relative $2u + u^2|cd|/|r|$, qui pour certaines valeurs de a, b, c, d est supérieure à 1.

Pour conclure que l'algorithme de Kahan est *toujours* très précis, on peut compléter cette analyse classique par la prise en compte de propriétés plus fines, selon le schéma suivant : si $cd \in \mathbb{F}$ ou $ad + \widehat{w} \in \mathbb{F}$, alors on vérifie aisément en invoquant (5.7) que $\widehat{r}$ est l'arrondi correct de la valeur exacte : $\widehat{r} = \mathrm{RN}(r)$; sinon, on peut remplacer la majoration $|e| \leqslant u|cd|$ vue plus haut par $|e| \leqslant (\beta - 1)|r|$. (Ce second point n'est pas trivial, mais découle de propriétés comme (5.3) et (5.6) ; voir [21].) On déduit dans les deux cas que si $r \neq 0$ alors $|\widehat{r} - r|/|r| = O(u)$ pour β fixé.

Algorithme de Cornea, Harrison, Tang. Pour conclure, notons qu'il existe une variante de l'algorithme de Kahan [10, p. 273] permettant de garantir que la valeur renvoyée pour $ab + cd$ est la même que celle renvoyée pour $cd + ab$. (Cette propriété peut être utile pour fournir une implantation de la multiplication de deux nombres complexes qui préserve la commutativité.) Nous renvoyons à [20] pour une analyse fine de cette variante, mêlant modèle standard, analyse en ufp et utilisation du théorème 5.2.2.

5.4 Encadrer les erreurs avec des intervalles

Motivation. L'analyse directe de l'erreur flottante *a posteriori* peut être automatisée, avec plus ou moins d'efforts et plus ou moins de succès. C'est en tout cas l'optique qui a présidé à la définition de l'arithmétique par intervalles et surtout de l'analyse par intervalles dans les années 1950-1960 [3]. Très rapidement, cette préoccupation s'est doublée du besoin de savoir prendre en compte d'autres erreurs : les incertitudes, de mesure ou autres, sur les données des calculs. L'arithmétique par intervalles incorpore de façon indifférenciée ces différentes sources d'erreur.

Avant de voir comment utiliser l'arithmétique par intervalles pour analyser les erreurs d'arrondi des calculs, il est important de définir cette arithmétique avec une approche purement mathématique, afin d'assurer le bien-fondé des définitions [4].

Cette section commence par une définition des intervalles, puis décrit comment calculer avec des intervalles et se termine par la propriété fondamentale, dite propriété d'inclusion, qui justifie l'intérêt de l'arithmétique par intervalles en général, et en particulier pour les majorations des erreurs dues à l'arithmétique flottante.

5.4.1 Arithmétique par intervalles : présentation

Le principe de l'arithmétique par intervalles est de manipuler et de calculer non avec des nombres, mais avec des intervalles.

Définition 5.4.1. *Un intervalle est un sous-ensemble connexe fermé de* $\mathbb{R}$.

3. Les origines historiques de l'arithmétique par intervalles étant sujettes à discussion, contentons-nous de citer Sunaga pour son mémoire de master en japonais, intitulé "Geometry of Numerals", en 1956 à l'Université de Tokyo, et Moore pour le livre [34] paru en 1966 reprenant ses travaux de thèse. Pour un historique détaillé, on peut consulter http://www.cs.utep.edu/interval-comp/, rubrique *Early Papers, by Others*, ainsi que l'article *The origin of interval arithmetic*, de Rump, présenté à SCAN 2016.

4. On s'appuiera pour cette partie sur les ouvrages introductifs de Moore [35], Moore, Kearfott et Cloud [36] ainsi que celui de Tucker [64].

Par exemple, $\emptyset$, $[-1,3]$, $]-\infty,2]$, $[5,+\infty[$ et $\mathbb{R}$ sont des intervalles. En revanche $]-1,3]$, $]0,+\infty[$ ou $[1,2]\cup[3,4]$ ne sont pas des intervalles : les deux premiers ne sont pas fermés et le dernier n'est pas connexe.

La norme IEEE 754 pour l'arithmétique flottante définit les infinis $+\infty$ et $-\infty$ et permet de calculer avec ces valeurs. Cependant les analyses d'erreur d'arrondi ne sont plus valides en présence de valeurs infinies. Nous supposerons désormais que toutes les valeurs manipulées sont finies.

On notera en gras l'intervalle : $\mathbf{x} = \{x \in \mathbf{x}\}$ et on notera ses extrémités par $\underline{x}$, $\bar{x}$ avec $\underline{x} \in \mathbb{R}$, $\bar{x} \in \mathbb{R}$: $\mathbf{x} = [\underline{x}, \bar{x}] = \{x : \underline{x} \leqslant x \leqslant \bar{x}\}$.

Définition 5.4.2. *Une opération n-aire sur les réels* op $: \mathbb{R}^n \to \mathbb{R}$*, s'étend en une opération n-aire sur les intervalles en utilisant la théorie des ensembles :*

$$\mathrm{op}(\mathbf{x}_1, \dots \mathbf{x}_n) = \mathrm{Hull}\{\mathrm{op}(x_1, \dots x_n),\ x_i \in \mathbf{x}_i \textit{ pour } 1 \leqslant i \leqslant n\},$$

où Hull désigne l'enveloppe convexe de l'ensemble. On ne considère que les opérandes $(x_i)_{1\leqslant i\leqslant n}$ appartenant au domaine de définition de op*.*

Cette définition théorique peut s'incarner dans des formules que l'on peut obtenir simplement, en utilisant la monotonie des opérateurs :

$$\begin{array}{lcl}
\mathbf{x}+\mathbf{y} & = & [\underline{x}+\underline{y}, \bar{x}+\bar{y}] \\
\mathbf{x}-\mathbf{y} & = & [\underline{x}-\bar{y}, \bar{x}-\underline{y}] \\
\mathbf{x}\cdot\mathbf{y} & = & [\min(\underline{x}\cdot\underline{y}, \underline{x}\cdot\bar{y}, \bar{x}\cdot\underline{y}, \bar{x}\cdot\bar{y}), \max(\underline{x}\cdot\underline{y}, \underline{x}\cdot\bar{y}, \bar{x}\cdot\underline{y}, \bar{x}\cdot\bar{y})] \\
\mathbf{x}/\mathbf{y} & = & [\min(\underline{x}/\underline{y}, \underline{x}/\bar{y}, \bar{x}/\underline{y}, \bar{x}/\bar{y}), \max(\underline{x}/\underline{y}, \underline{x}/\bar{y}, \bar{x}/\underline{y}, \bar{x}/\bar{y})] \text{ si } 0 \neq \mathbf{y} \\
\sqrt{\mathbf{x}} & = & \begin{cases} [\sqrt{\underline{x}}, \sqrt{\bar{x}}] & \text{si} \quad \underline{x} \geqslant 0, \\ [0, \sqrt{\bar{x}}] & \text{si} \quad \underline{x} \leqslant 0 \leqslant \bar{x}, \\ \emptyset & \text{si} \quad \bar{x} < 0. \end{cases}
\end{array}$$

Pour $\cdot$ et $/$, on décompose $\mathbf{x}$ et $\mathbf{y}$ selon le signe de leurs éléments et on utilise les monotonies partielles sur $\mathbf{x}\cap\mathbb{R}^-$, $\mathbf{x}\cap\mathbb{R}^+$ ... Toujours en utilisant les monotonies des opérations et des fonctions, on peut déterminer des formules pour calculer le logarithme, le sinus ou toute autre fonction d'un intervalle, elles sont simplement plus longues à écrire quand la fonction n'est pas monotone.

La propriété qui justifie l'intérêt de ces calculs est la suivante.

Propriété 5.4.3 (Théorème fondamental de l'arithmétique par intervalles, ou propriété d'inclusion, ou « Tu ne mentiras point »)**.** *Soit c un calcul portant sur n nombres réels $x_1, \dots x_n$, qui est donné par une expression arithmétique complètement parenthésée, ou encore par un DAG (directed acyclic graph). Remplacer dans cette représentation de c les nombres $x_1, \dots x_n$ par des intervalles $\mathbf{x}_1, \dots \mathbf{x}_n$:*

$c(\mathbf{x}_1, \ldots \mathbf{x}_n)$, puis évaluer les opérations comme indiqué précédemment, conduit à un résultat $\mathbf{y}$ *qui vérifie*

$$\mathbf{y} \supset \{c(x_1, \ldots x_n), x_i \in \mathbf{x}_i \text{ pour } i = 1 \text{ à } n\}.$$

En d'autres termes, le résultat d'un calcul mené en arithmétique par intervalles contient, inclut (d'où le nom de cette propriété) le résultat exact du calcul sur les réels. Ce résultat « ne ment pas » au sens où le résultat exact ne peut pas se trouver ailleurs.

5.4.2 Implantations et difficultés

Ces formules permettent d'implanter les différentes opérations et fonctions utiles, si on dispose d'une arithmétique exacte pour le calcul des extrémités. Sinon, afin de préserver la propriété d'inclusion, on calcule un résultat $\widehat{\mathbf{z}} = [\widehat{\underline{z}}, \widehat{\bar{z}}]$ qui contient le résultat $\mathbf{z} = [\underline{z}, \bar{z}]$, autrement dit il faut que $\widehat{\underline{z}} \leqslant \underline{z}$ et que $\bar{z} \leqslant \widehat{\bar{z}}$. C'est possible si on utilise l'arithmétique flottante, grâce aux arrondis dirigés : par exemple, si $\underline{x}, \bar{x}, \underline{y}$ et $\bar{y}$ sont des nombres flottants, pour calculer $\mathbf{x} + \mathbf{y} = [\underline{x}, \bar{x}] + [\underline{y}, \bar{y}] = [\underline{x} + \underline{y}, \bar{x} + \bar{y}]$ on retourne le plus petit intervalle (au sens de l'inclusion) qui contient le résultat exact, il s'agit de $[\mathrm{RD}(\underline{x} + \underline{y}), \mathrm{RU}(\bar{x} + \bar{y})]$.

Il semble que la présence des arrondis dirigés RD et RU dans la norme IEEE 754 pour l'arithmétique flottante ait été imposée par Kahan, afin de faciliter l'implantation de l'arithmétique par intervalles. Cependant, pendant longtemps les langages de programmation de haut niveau ne donnaient pas accès aisément aux modes d'arrondi (et ne l'offrent toujours pas de façon normalisée). Il fallait avoir recours à des appels à des routines en assembleur, ce qui n'était ni intuitif ni portable. La situation s'est améliorée au fil des années et, par exemple, en langage C les appels à `fegetround()` et `fesetround()` fonctionnent sur la majorité des architectures avec le fichier `fenv.h`.

Une autre difficulté liée à l'utilisation des modes d'arrondis dirigés concerne l'efficacité des programmes, ou plutôt leur inefficacité, avec des temps d'exécution multipliés par des facteurs allant de 10 à 50 ou même 1000, en comparaison des mêmes programmes utilisant uniquement l'arithmétique flottante. Une première explication est que chaque opération sur des intervalles nécessitant plusieurs opérations flottantes, le calcul avec des intervalles est intrinsèquement plus lent que le calcul flottant. Par exemple, comme il faut 2 opérations flottantes pour l'addition de deux intervalles, cette dernière est deux fois plus lente que l'addition flottante. Il faut même 8 opérations flottantes pour la multiplication d'intervalles puisqu'il faut effectuer les quatre produits en arrondi vers $-\infty$ pour calculer la borne

gauche et en arrondi vers $+\infty$ pour la borne droite, et donc 8 fois plus de temps que pour une multiplication flottante. Mais cela n'explique pas tout. Ce ralentissement est pour une grande part dû au fait que, sur les architectures classiques telles que IA32 et IA64 (plus connues sous le nom de x86), le mode d'arrondi est une information gérée par un drapeau d'état (*status flag*). Lorsque ce drapeau est modifié, toutes les instructions en cours d'exécution dans le pipeline sont arrêtées et recommencées avec le nouveau mode d'arrondi en vigueur. Si chaque opération en arithmétique par intervalles requiert au moins un changement de mode d'arrondi, plus aucun calcul ne peut être vectorisé, ce qui explique la perte d'efficacité d'un à deux ordres de grandeur. Nous avons décrit dans [47] d'autres problèmes liés aux changements de modes d'arrondi qui se posent lorsque les calculs sont parallèles, nous ne les aborderons pas ici. Ce qui justifie la perte du troisième ordre de grandeur est le remplacement des calculs flottants optimisés (pour l'utilisation des niveaux de cache etc.) – typiquement, des routines BLAS ou Lapack – par des calculs souvent moins optimisés.

Sur des architectures moins classiques, telles que les GPU ou la récente architecture Knight Landing d'Intel, ces problèmes ne devraient plus se poser puisque le mode d'arrondi est précisé dans le code de l'opération. Avec des implantations en CUDA pour GPU, on observe des accélérations, par rapport aux calculs sur CPU classiques, de 2 à 100 pour l'algorithme de Newton dans [2] et jusqu'à 1000 pour un calcul AXPY dans [8].

Difficultés d'utilisation : perte de propriétés algébriques. D'autres difficultés concernent l'écriture d'algorithmes utilisant l'arithmétique par intervalles et elles n'ont pas de solution simple et systématique. Il s'agit du fait que, comparée à l'arithmétique sur les nombres réels, l'arithmétique par intervalles « perd » certaines propriétés algébriques. Tout d'abord, la formule pour l'addition de deux intervalles $\mathbf{x}$ et $\mathbf{y}$:

$$\mathbf{x} + \mathbf{y} = [\underline{x}, \bar{x}] + [\underline{y}, \bar{y}] = [\underline{x} + \underline{y}, \bar{x} + \bar{y}]$$

permet de constater qu'en général un intervalle $\mathbf{x}$ n'a pas d'opposé : si $\underline{x} \neq \bar{x}$, il n'existe pas d'intervalle $\mathbf{y} = [\underline{y}, \bar{y}]$ tel que $\mathbf{x} + \mathbf{y} = [0, 0]$. En effet, si $\underline{x} \neq \bar{x}$, la largeur $w(\mathbf{x}) = \bar{x} - \underline{x}$ est strictement positive. Comme la largeur de $\mathbf{x} + \mathbf{y}$ est la somme des largeurs de $\mathbf{x}$ et $\mathbf{y}$: $w(\mathbf{x} + \mathbf{y}) = w([\underline{x} + \underline{y}, \bar{x} + \bar{y}]) = (\bar{x} - \underline{x}) + (\bar{y} - \underline{y}) = w(\mathbf{x}) + w(\mathbf{y})$, on a $w(\mathbf{x} + \mathbf{y}) > w(\mathbf{y}) \geqslant 0 = w([0, 0])$, on ne peut pas trouver $\mathbf{y}$ tel que $\mathbf{x} + \mathbf{y} = [0, 0]$. Toute addition ou soustraction ne fait qu'accroître la largeur des intervalles. Une conséquence est que l'on ne peut pas utiliser une formule mathématique qui repose sur une simplification de la forme « $x - x = 0$ » lorsque l'arithmétique utilisée est l'arithmétique par intervalles. Or les formules de Strassen pour

la multiplication rapide de matrices utilisent cette réécriture, tout comme les calculs de différences finies pour approcher des dérivées.

Pour la multiplication également, un intervalle $\mathbf{x} = [\underline{x}, \bar{x}] \not\ni 0$ n'a en général pas d'inverse : il n'existe pas toujours d'intervalle $\mathbf{y} = [\underline{y}, \bar{y}]$ tel que $\mathbf{x} \cdot \mathbf{y} = [\min(\underline{x} \cdot \underline{y}, \underline{x} \cdot \bar{y}, \bar{x} \cdot \underline{y}, \bar{x} \cdot \bar{y}), \max(\underline{x} \cdot \underline{y}, \underline{x} \cdot \bar{y}, \bar{x} \cdot \underline{y}, \bar{x} \cdot \bar{y})] = [1, 1]$. Comme $\mathbf{x} \not\ni 0$, $\underline{x}\bar{x} > \overline{0}$. Supposons $\underline{x} > 0$, le cas où $\bar{x} < 0$ est similaire. On peut s'attendre à ce que l'inverse $\mathbf{y}$ vérifie aussi $\underline{y} > 0$. La formule pour le produit $\mathbf{x} \cdot \mathbf{y}$ se simplifie dans ce cas en $\mathbf{x} \cdot \mathbf{y} = [\underline{x} \cdot \underline{y}, \bar{x} \cdot \bar{y}]$. Pour que $\mathbf{x} \cdot \mathbf{y}$ soit égal à $[1, 1]$, il faut donc que $\underline{y} = 1/\underline{x}$ et que $\bar{y} = 1/\bar{x}$. On a alors $\mathbf{y} = [1/\underline{x}, 1/\bar{x}]$ avec $1/\underline{x} \leqslant 1/\bar{x}$, ce qui n'est possible que si $\underline{x} = \bar{x}$ et $\underline{y} = \bar{y}$. Dans ce cas, $\mathbf{x}$ ne contient qu'un seul nombre réel. Comme pour l'addition, si $w(\mathbf{x}) > 0$ alors $\mathbf{x}$ n'a pas d'inverse.

Une dernière propriété algébrique importante qui n'est pas vérifiée par l'arithmétique par intervalles est la distributivité de la multiplication sur l'addition : on a simplement $\mathbf{x} \cdot (\mathbf{y} + \mathbf{z}) \subset \mathbf{x} \cdot \mathbf{y} + \mathbf{x} \cdot \mathbf{z}$. Il suffit d'un exemple pour s'en convaincre : $[1, 2] \cdot ([-5, -3] + [4, 7]) = [1, 2] \cdot [-1, 4] = [-2, 8]$ et $[1, 2] \cdot [-5, -3] + [1, 2] \cdot [4, 7] = [-10, -3] + [4, 14] = [-6, 11] \supset [-2, 8]$.

L'explication de ces pertes de propriétés est connue sous le nom de *décorrélation des variables* ou en anglais *variable dependency*. Revenons à la définition d'une opération, par exemple $\mathbf{x} - \mathbf{y} = \{x - y,\ x \in \mathbf{x},\ y \in \mathbf{y}\}$. Si $\mathbf{y} = \mathbf{x}$, on a donc $\mathbf{x} - \mathbf{x} = \{x - y,\ x \in \mathbf{x},\ y \in \mathbf{x}\}$: cet ensemble contient 0, obtenu dans le cas où $x = y$, mais aussi d'autres éléments, obtenus quand $x \neq y$. Si on avait en tête $x - x = 0$, c'est-à-dire le fait que les deux occurrences de x sont identiques, cette information est perdue lors de l'écriture $\mathbf{x} - \mathbf{x}$: les deux occurrences sont décorrélées.

L'absence de ces propriétés algébriques signifie que l'écriture des formules, des algorithmes utilisant l'arithmétique par intervalles est délicate et demande d'utiliser $x \cdot (y + z)$ plutôt que $x \cdot y + x \cdot z$ ou d'éviter de réécrire 0 en $x - x$. Les règles à suivre sont en résumé les suivantes : il est préférable de n'introduire des intervalles dans les formules que le plus tard possible, après avoir réalisé toutes les manipulations algébriques souhaitées sur les expressions portant sur des nombres réels, et le plus parcimonieusement possible, en conservant le plus possible des variables réelles et non intervalles. Différentes variantes de l'arithmétique par intervalles, telles que l'arithmétique affine ou les modèles de Taylor, ont été développées pour pallier ces problèmes de décorrélation des variables. Elles sortent du cadre strict de l'analyse a posteriori et automatique des erreurs d'arrondi, puisqu'elles font appel à des reformulations des expressions évaluées et elles entraînent donc d'autres erreurs d'arrondi que celles des expressions originales. Elles seront brièvement évoquées en fin de ce chapitre.

5.4.3 Augmenter la précision de calcul

Analyse a posteriori. Dans cette section, on dispose d'une expression et d'une arithmétique flottante pour l'évaluer. L'approche choisie consiste à remplacer, dans cette expression, les données par des intervalles, à bornes flottantes, les contenant, et les opérations par des opérations sur ces intervalles, implantées en utilisant l'arithmétique flottante. Le résultat $\widehat{\mathbf{r}}$ de ce calcul par intervalles est un intervalle à bornes flottantes qui contient le résultat exact r et les erreurs d'arrondi commises lors des calculs : l'intervalle $\widehat{\mathbf{r}}$ permet d'encadrer les erreurs d'arrondi commises lors de l'évaluation de l'expression. Plus précisément la largeur $w(\widehat{\mathbf{r}})$ majore l'erreur absolue commise sur r et on s'intéresse donc à cette largeur. Comme les erreurs d'arrondi commises lors de l'évaluation des bornes, pour chaque opération intermédiaire, sont également prises en compte, $\widehat{\mathbf{r}}$ encadre, outre les erreurs d'arrondi commises lors du calcul de r, toutes ces erreurs d'arrondi sur les bornes. La largeur de $\widehat{\mathbf{r}}$ est donc fréquemment pessimiste – on parle de *surencadrement*.

Tout d'abord, quelle est la largeur du plus petit (pour l'inclusion) intervalle $\widehat{\mathbf{t}}$ à bornes flottantes contenant un nombre réel $t \in \mathbb{R}$? On a vu que $\mathrm{RD}(t)$ et $\mathrm{RU}(t)$ définissent l'unique paire d'éléments consécutifs de $\mathbb{F}$ (ou égaux si $t \in \mathbb{F}$) telle que $t \in [\mathrm{RD}(t), \mathrm{RU}(t)]$, on peut donc utiliser (5.4) pour majorer la largeur de $\widehat{\mathbf{t}}$ par $w(\widehat{\mathbf{t}}) = \mathrm{RU}(t) - \mathrm{RD}(t) \leqslant 2u\beta^e \leqslant 2u|t|$. La largeur des intervalles qui sont utilisés comme données est donc au pire proportionnelle à u et à la valeur absolue des données réelles.

Si maintenant on calcule $\widehat{\mathbf{r}} = \widehat{\mathrm{op}}(\mathbf{x}_1, \ldots \mathbf{x}_n)$, on a $\widehat{\mathbf{r}} = [\mathrm{RD}(r_1), \mathrm{RU}(r_2)]$, où r_1 et r_2 sont donnés par les formules de la section 5.4.1, donc on ajoute à la largeur du résultat exact $[r_1, r_2]$ une quantité que l'on peut majorer comme en section 5.2.2. Par exemple, si $\mathbf{x} \geqslant 0$ et $\mathbf{y} \geqslant 0$, c'est-à-dire que $\underline{x} \geqslant 0$ et $\underline{y} \geqslant 0$, et si $\mathbf{r} = \mathbf{x} + \mathbf{y}$ et $\widehat{\mathbf{r}} = \widehat{\mathbf{x} + \mathbf{y}} = [\mathrm{RD}(\underline{x} + \underline{y}), \mathrm{RU}(\bar{x} + \bar{y})]$, alors

$$\begin{aligned} w(\widehat{\mathbf{r}}) = \mathrm{RU}(\bar{x} + \bar{y}) - \mathrm{RD}(\underline{x} + \underline{y}) &\leqslant (1 + 2u)(\bar{x} + \bar{y}) - (1 - 2u)(\underline{x} + \underline{y}) \\ &\leqslant w(\mathbf{x} + \mathbf{y}) + 4u\tfrac{(\underline{x}+\underline{y})+(\bar{x}+\bar{y})}{2}. \end{aligned}$$

On notera $\mathrm{mid}(\mathbf{r})$ le milieu de $\mathbf{r}$: $\mathrm{mid}(\mathbf{r}) = (\underline{r} + \bar{r})/2$. La largeur relative du résultat de l'addition, $w(\widehat{\mathbf{r}})/\mathrm{mid}(\mathbf{r})$, est augmentée de $4u$ comparée à la largeur relative du résultat exact. Il en va de même pour les autres opérations. On voit apparaître un analogue de l'équation (5.11) de la modélisation standard pour le calcul par intervalles, avec un accroissement de la largeur relative des intervalles calculés d'un facteur $4u$.

Plus généralement, si l'expression r et chacune de ses sous-expressions sont lipschitziennes en les variables, la surestimation du résultat calculé

$w(\widehat{\mathbf{r}}) - w(\mathbf{r})$ est au pire proportionnelle à u : lorsque la précision de calcul p augmente, la largeur du résultat diminue. On trouve la preuve, par induction sur le DAG qui définit r, dans Neumaier [40, Theorem 2.1.5].

Quelques exemples. Illustrons ce résultat sur deux exemples, à l'aide de la bibliothèque MPFI [46] (qui calcule en base 2). Commençons par un exemple sans surprise : « vérifions numériquement » la formule de Machin :

$$\frac{\pi}{4} = 4 * \arctan(\frac{1}{5}) - \arctan(\frac{1}{239})$$

en calculant la différence entre $\pi/4$ et le terme de droite. On observe bien une largeur du résultat proportionnelle à l'unité d'erreur d'arrondi.

```
Enter the computing precision: 20
Difference = [-1.9073487e-6,1.9073487e-6]

Enter the computing precision: 40
Difference = [-1.8189894035459e-12,1.8189894035459e-12]

Enter the computing precision: 80
Difference = [-1.6543612251060553497428174e-24,1.6543612251060553497428174e-24]
```

Voici un autre exemple, la formule de Rump, construite pour provoquer des éliminations catastrophiques (cf. section 5.2.4) « en cascade », pour différentes valeurs d'exposants, tant que les calculs sont menés avec moins de 122 bits. On observe bien des encadrements très larges tant que la précision de calcul est insuffisante, puis très fins ensuite.

```
Rump's formula:
(333+3/4)*b^6 + a^2*(11*a^2*b^2-b^6-121*b^4-2) + 11/2*b^8 + a/2b
with a = 77617.0 and b = 33096.0
   single precision : 136652528711380592394066862325261926 4.000000
   double precision : 136652461123916627060868321645900595 2.000000

Enter the computing precision: 24
Eval. =  [-8.23972891e30,8.23972951e30]

Enter the computing precision: 53
Eval. =  [-5.9029581035870566e21,4.7223664828696463e21]

Enter the computing precision: 121
Eval. =  [-2.8273960599468213681411650954798162924,
           1.1726039400531786318588349045201837084]

Enter the computing precision: 122
Eval. =  [-8.2739605994682136814116509547981629201e-1,
          -8.2739605994682136814116509547981629162e-1]
```

Utiliser l'arithmétique par intervalles en faisant varier la précision de calcul ne demande pas une grande expertise, pas beaucoup de temps de développement et fournit des résultats intéressants, bien qu'inefficacement. C'est donc une approche assez automatique qui mérite d'être essayée.

Dans Gappa [5] [11, 5], l'arithmétique par intervalles en précision arbitraire est utilisée pour majorer les erreurs d'arrondi. Très schématiquement, elle intervient après une passe de réécriture des erreurs sous une forme facilement exploitable, en faisant apparaître explicitement les termes de la forme $x - \hat{x}$ où x est la valeur exacte et $\hat{x}$ la valeur approchée. La dernière phase de Gappa consiste à produire une preuve formelle, qui peut être vérifiée par l'assistant de preuve Coq, de la borne produite.

5.4.4 Pour plus d'efficacité : représentation par centre et rayon

Représentation mid-rad. Dans les expériences numériques ci-dessus, on aimerait que la dernière valeur soit affichée comme $-0.82739605994682136814116509547981629201 \pm 2.05 \cdot 10^{-37}$, c'est-à-dire qu'elle soit représentée par son *milieu* (également appelé *centre*) et son *rayon*, ce qui s'appelle en anglais une représentation *mid-rad* pour *midpoint-radius*. Plusieurs bibliothèques utilisent cette représentation, telles que IntLab qui utilise deux flottants [50], ou ARB [24] et Mathemagix [30] qui stockent le centre avec une grande précision et le rayon avec une précision fixée.

Formules pour les opérations arithmétiques, adaptées à la représentation mid-rad. Notons $\mathbf{x} =< x_m, x_r >$ un intervalle $\mathbf{x}$ représenté par son milieu x_m et son rayon x_r (avec $x_r \geqslant 0$), $\mathbf{y} =< y_m, y_r >$ et $\mathbf{z} =< z_m, z_r >= \mathbf{x}\,\mathrm{op}\,\mathbf{y}$. On a par exemple

$$\begin{aligned} \mathbf{x}+\mathbf{y} &= < x_m + y_m, x_r + y_r > \\ \mathbf{x}-\mathbf{y} &= < x_m - y_m, x_r + y_r > . \end{aligned}$$

Pour implanter ces formules en arithmétique flottante, on calcule le milieu en arrondi au plus près. Le rayon est calculé en arrondi vers $+\infty$ afin de surestimer le rayon exact et de préserver la propriété d'inclusion ; de plus, il incorpore une majoration de l'erreur d'arrondi commise en calculant le milieu. Pour cela, ARB et Mathemagix ajoutent $u|z_m|$ au rayon.

Pour la multiplication, on trouve une formule exacte dans [40, Property 1.6.5]. Elle est assez compliquée et requiert au total 9 multiplications entre des réels. Très souvent, le rayon est surestimé à l'aide d'une formule plus simple. Dès 1999, Rump a proposé dans [49] de calculer z_m et z_r comme

$$z_m = x_m \cdot y_m \quad \text{et} \quad z_r = (|x_m| + x_r) \cdot y_r + x_r \cdot |y_m|.$$

5. http://gappa.gforge.inria.fr/

Nguyen dans sa thèse [41] en 2011, Rump dans [51], Théveny dans sa thèse [61] en 2014, pour n'en citer que quelques-uns, proposent d'autres formules faisant intervenir de 2 à 7 multiplications entre des réels, et offrant des surestimations (en arithmétique exacte) plus ou moins larges.

Pour l'implantation flottante, les erreurs d'arrondi sont prises en compte dans le calcul du rayon : $z_m = \mathrm{RN}(x_m \cdot y_m)$ et $z_r = \mathrm{RU}((|x_m| + x_r) \cdot y_r + x_r \cdot |y_m| + u \cdot |z_m|)$.

Efficacité des calculs. De telles formules sont particulièrement intéressantes lorsque l'on multiplie des matrices dont les coefficients sont des intervalles $\mathbf{A} \cdot \mathbf{B} =< A_m, A_r > \cdot < B_m, B_r >$; ces formules s'appliquent en remplaçant les nombres réels par des matrices et en appliquant la valeur absolue à chaque élément d'une matrice. On peut alors utiliser des routines optimisées et obtenir des produits efficaces pour les matrices à coefficients intervalles : le facteur de ralentissement dans [61] est de 3, bien loin des 1 à 3 ordres de grandeur mentionnés plus haut.

Cette approche est particulièrement intéressante pour les calculs parallèles. Si on stocke les rayons en simple précision plutôt qu'en double, chaque opération sur les rayons est très rapide et chaque mouvement de donnée est beaucoup moins gourmand, en temps et en énergie. Enfin, chaque valeur numérique, un centre et un rayon donc, donne lieu à plusieurs opérations arithmétiques avant d'être à nouveau stockée : l'intensité numérique du calcul est augmentée. Effectuer des calculs garantis pourrait devenir beaucoup moins pénalisant et donc plus attractif qu'à l'heure actuelle… mais il reste encore des efforts à faire pour optimiser les mouvements de données et l'intensité numérique.

Influence des erreurs d'arrondi. Le rayon du résultat est un encadrement des erreurs d'arrondi accumulées lors du calcul du centre et de son propre calcul. Dans les calculs en représentation mid-rad, quelle est la part relative de ces erreurs d'arrondi et de la largeur des intervalles en entrée ? Nous présentons ici quelques-uns des résultats de Théveny [61].

La figure 5.1 est tirée de [61, Fig. 4.14]. Le calcul considéré est un produit de matrices $\mathbf{A} \cdot \mathbf{B}$ de dimensions 128×128 dont les coefficients sont des intervalles, choisies de sorte que l'incertitude relative sur les coefficients de $\mathbf{A}$, $\mathbf{A}_{i,j} =< A_{mi,j}, A_{ri,j} >$ et de $\mathbf{B}$ soit majorée par e :

$$\frac{A_{ri,j}}{|A_{mi,j}|} \leqslant e, \quad \frac{B_{ri,j}}{|B_{mi,j}|} \leqslant e,$$

où la valeur e est atteinte. C'est cette valeur e qui figure en abscisse.

En ordonnée se trouve la même quantité pour $\mathbf{C}$ résultant du calcul $\mathbf{A} \cdot \mathbf{B}$ par deux algorithmes différents : il faut 3 produits de matrices pour

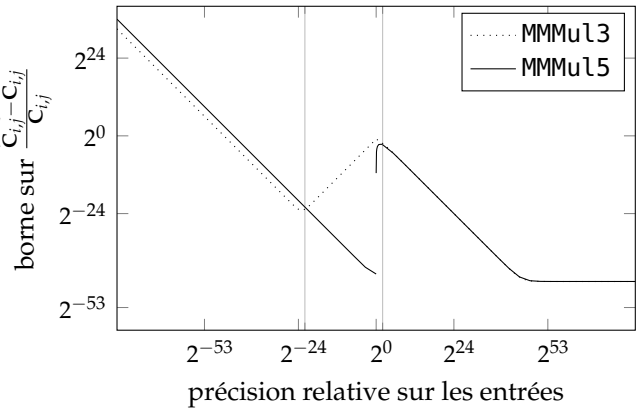

FIGURE 5.1 – *Comparaison des encadrements des largeurs relatives des rayons des matrices calculées par* `MMul3` *et* `MMul5`.

`MMul3` et 5 pour `MMul5`. Avec une arithmétique exacte, une étude théorique a établi que `MMul5` est plus précis que `MMul3`. Expérimentalement, avec une arithmétique flottante double précision, on observe l'inverse quand les intervalles de **A** et **B** sont très fins : les rayons des éléments de **C** correspondent aux majorations des erreurs d'arrondi. Effectuant moins d'opérations, `MMul3` donne lieu à des erreurs d'arrondi plus faibles et donc à une courbe plus basse sur la partie gauche de la figure. Quand e croît, l'influence de la largeur des rayons des données prédomine et la plus faible surestimation du rayon, en arithmétique exacte, de `MMul5` devient manifeste. Les rayons cessent de correspondre aux erreurs d'arrondi des calculs.

Dans d'autres expériences, tirées de [61], les rayons des données sont fixés à un ulp des centres correspondants. Une grande majorité des rayons des résultats restent proches de l'ulp du coefficient correspondant : les erreurs d'arrondi initiales n'ont été que peu amplifiées par les calculs. Les autres sont proches de la borne théorique, obtenue a priori, et cela se produit pour les calculs dits *mal conditionnés* (évoqués en section 5.2.4), c'est-à-dire pour lesquels l'analyse prédit une large amplification des erreurs initiales.

5.4.5 Autres approches : arithmétique affine, modèles de Taylor

Terminons par quelques variantes de l'arithmétique par intervalles, qui réduisent le problème de décorrélation des variables. Ces variantes sortent du cadre strict de ce chapitre et sont mentionnées parce qu'elles sont employées en situation réelle ; voir la littérature citée ci-dessous et [57, 60] pour des exemples d'applications. Pour chaque opération, le temps d'exé-

cution est de 2 à 3 ordres de grandeur supérieur au temps du calcul flottant, mais le gain en précision, et donc la réduction des calculs nécessaires à l'obtention d'un résultat garanti et précis, justifient ce surcoût.

Arithmétique affine. Cette arithmétique a été proposée par Comba et Stolfi en 1993 [9] et détaillée dans [59]. Le principe qui prévaut est de remplacer chaque variable x (que ce soit une variable d'entrée ou un résultat de calcul intermédiaire) par une combinaison affine $x_0 + \sum_{i=1}^{n} x_i\varepsilon_i$, où les x_i sont des réels, et chaque ε_i est un symbole de bruit, indépendant des autres $\varepsilon_j, j \neq i$, et $\varepsilon_i \in [-1, 1]$. Voyons comment définir les opérations. Pour l'addition ou la soustraction de $x = x_0 + \sum_{i=1}^{n} x_i\varepsilon_i$ et $y = y_0 + \sum_{i=1}^{n} y_i\varepsilon_i$, on a :

$$z = x \pm y = x_0 \pm y_0 + \sum_{i=1}^{n} (x_i \pm y_i)\varepsilon_i : \left\{ \begin{array}{lcl} z_0 & = & x_0 \pm y_0, \\ z_i & = & x_i \pm y_i \text{ pour } i = 1 \dots n. \end{array} \right.$$

Conserver une expression symbolique permet par exemple de remédier au problème de l'expression $x - x$ qui s'évalue ainsi en 0.

Cela se complique pour la multiplication de deux variables : des termes de la forme $\varepsilon_i\varepsilon_j$ apparaissent, ils sont transformés en un nouveau terme $z_{n+1}\varepsilon_{n+1}$ qui nécessite la création d'une variable ε_{n+1}. Le même problème se pose pour la division et toute fonction non linéaire : $\sqrt{}$, exp, tan... Pour ces opérations, de nombreuses variantes existent, qui visent à limiter le nombre de variables créées lors d'un calcul et aussi à incorporer les erreurs d'arrondi commises sur chaque coefficient z_i. Citons les implantations en MATLAB [52] ou en C++ dans IBEX (cf. http://www.ibex-lib.org/).

Une application phare de l'arithmétique affine et d'autres variantes plus avancées (utilisant des zonotopes) est le logiciel Fluctuat, décrit par Putot dans [44] et Martel dans [32]. En quelques mots, *Fluctuat is a static analyzer by abstract interpretation, relying on the general-purpose zonotopic abstract domains, and dedicated to the analysis of numerical properties of programs.* L'arithmétique affine permet d'encadrer les valeurs des résultats, les erreurs d'arrondi et ainsi de garder une trace des opérations où elles se sont produites, afin d'identifier à quel moment elles posent problème.

Modèles polynomiaux, modèles de Taylor. Un tel modèle utilise des développements polynomiaux (et non affines) en les variables d'entrée, avec une seule variable pour chaque variable d'entrée et en fixant à l'avance le degré maximal des polynômes utilisés. Les implantations accessibles sont peu nombreuses : l'implantation la plus ancienne est due à Eckmann *et al.* [13], citons également COSY [45]. Enfin, on trouve des variations dans le choix des bases pour les approximations polynomiales : Bernstein chez Nataraj et Kotecha [38] pour l'obtention simple de bornes, Tchebychev chez Joldes [25, Chap. 4] pour la qualité de l'approximation polynomiale.

5.5 Notes bibliographiques

En ce qui concerne l'arithmétique à virgule flottante, nous renvoyons à [37] pour une présentation plus complète, notamment des détails de la norme IEEE 754. Des exemples d'implantation de cette norme figurent dans les thèses de Brunie [6], Lauter [29] et Revy [48], pour différents contextes.

Concernant l'analyse inverse, une référence reste le livre de Higham [16] ; le lien entre erreur inverse et erreur directe via la notion de conditionnement est présenté dans [16, §1] et [28].

Dans les sections 5.2 et 5.3, l'accent a été mis sur l'arrondi « au plus proche ». Il existe cependant quelques résultats récents visant à étendre ces résultats aux arrondis dirigés [4, 43]. De même, nous avons supposé que tous les calculs se font en l'absence de dépassement de capacité, c'est-a-dire qu'ils ne génèrent ni *underflow*, ni *overflow*. La prise en compte de ces exceptions, souvent délicate, a néanmoins fait l'objet de travaux récents, comme par exemple [3]. Dans tous les cas, la vérification formelle de l'ensemble de ces analyses, telle que présentée dans [5], est toujours d'actualité.

En ce qui concerne l'arithmétique par intervalles, une lecture complémentaire qui s'impose est celle de la récente norme IEEE 1788-2015 [19]. Les implantations de cette norme sont en cours, avec la bibliothèque `libieee1788` [39] en C++, qui repose sur une arithmétique flottante à précision arbitraire, ou la bibliothèque en Octave [15].

Il existe aussi une branche entière de l'analyse numérique dévolue au calcul avec des intervalles, qui permet d'obtenir des résultats inaccessibles lorsque l'on calcule avec des réels. Citons notamment la méthode de Newton pour encadrer tous les zéros d'une fonction f sur un intervalle donné $\mathbf{x}_0$, et ce souvent avec une preuve de leur existence et de leur unicité, ou a contrario qui fournit une preuve de l'absence de zéros. L'arithmétique par intervalles permet également, grâce à son caractère ensembliste, de développer des algorithmes d'optimisation globale, on trouvera une introduction dans [14]. Enfin, elle est utilisée avec succès pour l'intégration garantie d'équations différentielles ordinaires, par exemple pour prouver que le système de Lorenz admet un attracteur [63]. Les variantes (arithmétique affine, modèles polynomiaux) de l'arithmétique par intervalles sont employées dans ce cas, comme par exemple l'arithmétique affine dans [1].

Remerciements. Ce travail a été financé en partie par l'Agence Nationale de la Recherche (projet ANR-13-INSE-0007 *MetaLibm*).

Bibliographie

[1] J. Alexandre dit Sandretto and A. Chapoutot. Validated Explicit and Implicit Runge-Kutta Methods. *Reliable Computing*, 22 :78–103, 2016.

[2] P.-D. Beck and M. Nehmeier. Parallel Interval Newton Method on CUDA. In *International Workshop on Applied Parallel Computing*, pages 454–464. Springer, 2012.

[3] S. Boldo. Formal verification of programs computing the floating-point average. In M. Butler, S. Conchon, and F. Zaïdi, editors, *Proc. 17th International Conference on Formal Engineering Methods*, volume 9407 of *Lecture Notes in Computer Science*, pages 17–32, Paris, France, 2015. Springer International Publishing.

[4] S. Boldo, S. Graillat, and J.-M. Muller. On the robustness of the 2Sum and Fast2Sum algorithms, 2016. Preprint available at https ://hal-ens-lyon.archives-ouvertes.fr/ensl-01310023.

[5] S. Boldo and G. Melquiond. Arithmétique des ordinateurs et preuves formelles. In P. Langlois, editor, *Informatique Mathématique : une photographie en 2013*, pages 189–220. Presses Universitaires de Perpignan, 2013.

[6] N. Brunie. *Contributions to Computer Arithmetic and Applications to Embedded Systems*. PhD thesis, École normale supérieure de Lyon, Lyon, France, mai 2014. Accessible depuis https://tel.archives-ouvertes.fr/tel-01078204.

[7] P. Bürgisser and F. Cucker. *Condition*. Springer-Verlag Berlin Heidelberg, 2013.

[8] S. Collange, M. Daumas, and D. Defour. *GPU Computing Gems Jade Edition*, chapter Interval arithmetic in CUDA, pages 99–107. Morgan Kaufmann, 2011.

[9] J. L. D. Comba and J. Stolfi. Affine arithmetic and its applications to computer graphics. In *Proceedings of SIBGRAPI'93 - VI Simpósio Brasileiro de Computação Gráfica e Processamento de Imagens*, pages 9–18, 1993.

[10] M. Cornea, J. Harrison, and P. T. P. Tang. *Scientific Computing on Itanium®-based Systems*. Intel Press, Hillsboro, OR, USA, 2002.

[11] M. Daumas and G. Melquiond. Certification of bounds on expressions involving rounded operators. *Transactions on Mathematical Software*, 37(1) :1–20, 2010.

[12] T. J. Dekker. A floating-point technique for extending the available precision. *Numer. Math.*, 18 :224–242, 1971.

[13] J.-P. Eckmann, A. Malaspinas, and S. O. Kamphorst. *A Software Tool for Analysis in Function Spaces*, pages 147–167. Springer, 1991.

[14] E. Hansen and G. Walster. *Global optimization using interval analysis (2nd ed.)*. Marcel Dekker, 2003.

[15] O. Heimlich. Interval arithmetic in GNU Octave. In *SWIM 2016 : Summer Workshop on Interval Methods*, 2016.

[16] N. J. Higham. *Accuracy and Stability of Numerical Algorithms*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, second edition, 2002.

[17] IEEE Computer Society. *Symposium on Computer Arithmetic*, 1969–2016.

[18] IEEE Computer Society. *IEEE Standard for Floating-Point Arithmetic, IEEE Standard 754-2008*. IEEE Computer Society, New York, Aug. 2008.

[19] IEEE Computer Society. *1788-2015 - IEEE Standard for Interval Arithmetic*. IEEE Computer Society, New York, June 2015.

[20] C.-P. Jeannerod. A radix-independent error analysis of the Cornea-Harrison-Tang method. *ACM Trans. Math. Software*, 42(3) :19 :1–19 :20, 2016.

[21] C.-P. Jeannerod, N. Louvet, and J.-M. Muller. Further analysis of Kahan's algorithm for the accurate computation of 2×2 determinants. *Math. Comp.*, 82(284) :2245–2264, 2013.

[22] C.-P. Jeannerod and S. M. Rump. Improved error bounds for inner products in floating-point arithmetic. *SIAM J. Matrix Anal. Appl.*, 34(2) :338–344, 2013.

[23] C.-P. Jeannerod and S. M. Rump. On relative errors of floating-point operations : optimal bounds and applications, 2014. Preprint available at https ://hal.inria.fr/hal-00934443.

[24] F. Johansson. Arb : a C library for ball arithmetic. *ACM Communications in Computer Algebra*, 47(4) :166–169, 2013.

[25] M. Joldes. *Rigorous Polynomial Approximations and Applications*. PhD thesis, École Normale Supérieure de Lyon, 2011.

[26] W. Kahan. Further remarks on reducing truncation errors. *Comm. ACM*, 8(1) :40, 1965.

[27] D. E. Knuth. *The Art of Computer Programming, Volume 2, Seminumerical Algorithms*. Addison-Wesley, Reading, MA, USA, third edition, 1998.

[28] P. Langlois. *Outils d'analyse numérique pour l'automatique – Chapitre 1*, pages 19–52. Traité IC2 dirigé par Alain Barraud. Hermès Science, 2002.

[29] C. Q. Lauter. *Arrondi correct de fonctions mathématiques : fonctions univariées et bivariées, certification et automatisation*. PhD thesis, École normale supérieure de Lyon, Lyon, France, octobre 2008. Accessible depuis http://www.christoph-lauter.org/.

[30] G. Lecerf and J. van der Hoeven. Evaluating Straight-Line Programs over Balls. In *23rd IEEE Symposium on Computer Arithmetic*, pages 142–149, 2016.

[31] N. Louvet. *Algorithmes compensés en arithmétique flottante : précision, validation, performances*. PhD thesis, Université de Perpignan, Perpignan, France, Nov. 2007.

[32] M. Martel. Interprétation abstraite pour la précision numérique. In P. Langlois, editor, *Informatique Mathématique : une photographie en 2013*, pages 145–185. Presses Universitaires de Perpignan, 2013.

[33] O. Møller. Quasi double-precision in floating point addition. *BIT*, 5 :37–50, 1965.

[34] R. Moore. *Interval analysis*. Prentice Hall, 1966.

[35] R. Moore. *Methods and applications of interval analysis*. SIAM Studies in Applied Mathematics, 1979.

[36] R. Moore, R. Kearfott, and M. Cloud. *Introduction to Interval Analysis*. SIAM, 2009.

[37] J.-M. Muller, N. Brisebarre, F. de Dinechin, C.-P. Jeannerod, V. Lefèvre, G. Melquiond, N. Revol, D. Stehlé, and S. Torres. *Handbook of Floating-Point Arithmetic*. Birkhäuser Boston, 2010.

[38] P. Nataraj and K. Kotecha. Higher Order Convergence for Multidimensional Functions with a New Taylor-Bernstein Form as Inclusion Function. *Reliable Computing*, 9 :185–203, 2003.

[39] M. Nehmeier, J. Wolff von Gudenberg, and J. Pryce. On Implementing the IEEE Interval Standard P1788. In *FEMTEC 2013 : 4th International Congress on Computational Engineering and Sciences*, page 103, 2013.

[40] A. Neumaier. *Interval methods for systems of equations*. Cambridge University Press, 1990.

[41] H. D. Nguyen. *Efficient algorithms for verified scientific computing : Numerical linear algebra using interval arithmetic*. Phd thesis, Ecole normale supérieure de lyon - ENS LYON, Jan. 2011.

[42] T. Ogita, S. M. Rump, and S. Oishi. Accurate sum and dot product. *SIAM J. Sci. Comput.*, 26(6) :1955–1988, 2005.

[43] K. Ozaki, T. Ogita, F. Bünger, and S. Oishi. Accelerating interval matrix multiplication by mixed precision arithmetic. *Nonlinear Theory and Its Applications, IEICE*, 6(3) :364–376, 2015.

[44] S. Putot. *HDR : Static Analysis of Numerical Programs and Systems*. PhD thesis, Université de Paris-Sud, 2012.

[45] N. Revol, K. Makino, and M. Berz. Taylor models and floating-point arithmetic : proof that arithmetic operations are validated in COSY. *Journal of Logic and Algebraic Programming*, 64 :135–154, 2005.

[46] N. Revol and F. Rouillier. Motivations for an Arbitrary Precision Interval Arithmetic and the MPFI Library. *Reliable Computing*, 11(4) :275–290, 2005.

[47] N. R. Revol and P. Théveny. Numerical reproducibility and parallel computations : Issues for interval algorithms. *IEEE Transactions on Computers*, 63(8) :1915–1924, 2014.

[48] G. Revy. *Implementation of binary floating-point arithmetic on embedded integer processors – Polynomial evaluation-based algorithms and certified code generation.* PhD thesis, École normale supérieure de Lyon, Lyon, France, décembre 2009. Accessible depuis https://tel.archives-ouvertes.fr/tel-00469661/.

[49] S. Rump. Fast and parallel interval arithmetic. *BIT*, 39(3) :534–554, 1999.

[50] S. Rump. INTLAB - INTerval LABoratory. In T. Csendes, editor, *Developments in Reliable Computing*, pages 77–104. Kluwer Academic Publishers, Dordrecht, 1999. http://www.ti3.tuhh.de/rump/.

[51] S. Rump. Fast interval matrix multiplication. *Numerical Algorithms*, 1(61) :1–34, 2012.

[52] S. Rump and M. Kashiwagi. Implementation and improvements of affine arithmetic. *Nonlinear Theory and Its Applications, IEICE*, 6(3) :341–359, 2015.

[53] S. M. Rump. Error estimation of floating-point summation and dot product. *BIT*, 52(1) :201–220, 2012.

[54] S. M. Rump, F. Bünger, and C.-P. Jeannerod. Improved error bounds for floating-point products and Horner's scheme. *BIT*, 56(1) :293–307, 2016.

[55] S. M. Rump and C.-P. Jeannerod. Improved backward error bounds for LU and Cholesky factorizations. *SIAM J. Matrix Anal. Appl.*, 35(2) :684–698, 2014.

[56] S. M. Rump, T. Ogita, and S. Oishi. Accurate floating-point summation, Part I : Faithful rounding. *SIAM J. Sci. Comput.*, 31(1) :189–224, 2008.

[57] SCAN. *International Symposium on Scientific Computing, Computer Arithmetic and Validated Numerics*, 1989–2016.

[58] P. H. Sterbenz. *Floating-Point Computation*. Prentice-Hall, 1974.

[59] J. Stolfi and L. de Figueiredo. *Self-Validated Numerical Methods and Applications*. Monograph for 21st Brazilian Mathematics Colloquium, Rio de Janeiro, Brazil, 1997.

[60] SWIM. *Summer Workshop on Interval Methods*, 2008–2016.

[61] P. Théveny. *Numerical Quality and High Performance in Interval Linear Algebra on Multi-Core Processors*. PhD thesis, Université de Lyon, École Normale Supérieure de Lyon, 2014.

[62] L. N. Trefethen. Computing numerically with functions instead of numbers. *Math. Comput. Sci.*, 1(1) :9–19, 2007.

[63] W. Tucker. The Lorenz attractor exists. *Comptes Rendus de l'Académie des Sciences-Series I-Mathematics*, 328(12) :1197–1202, 1999.

[64] W. Tucker. *Validated Numerics – A Short Introduction to Rigorous Computations*. Princeton University Press, 2011.

[65] J. H. Wilkinson. Error analysis of floating-point computation. *Numer. Math.*, 2 :319–340, 1960.